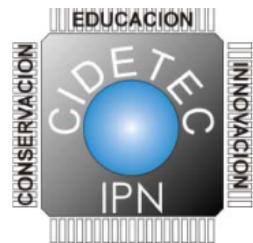




INSTITUTO POLITÉCNICO
NACIONAL
CENTRO DE INNOVACIÓN Y DESARROLLO
TECNOLÓGICO EN CÓMPUTO



**DESARROLLO DE UN SISTEMA QUE REPRESENTA OBJETOS
CON GEOMETRÍAS SIMPLES USANDO ÁLGEBRA
GEOMÉTRICA CONFORME**

T E S I S

Que para obtener el grado de:
Maestriá en Tecnología de Cómputo

Presenta:
Carlos Francisco Villaseñor Altamirano

Directores de la tesis:
M. en C. Marco Antonio Valencia Reyes
Dr. Gabriel Sepúlveda Cervantes

RESUMEN

En este trabajo se presenta una propuesta para el desarrollo de una interfaz que permita la interacción de objetos reales con un mundo virtual, usando un clasificador objetos son representados por geometrías simples. Modificando el proceso de clasificación usando álgebra geométrica conforme (AGC) se plantea mejorar el proceso.

Con el sensor Kinect se obtienen las nubes de puntos de los objetos, se realiza la estimación de parámetros para los modelos de la esfera, el plano y el cilindro, usando el método RANSAC, y posteriormente seleccionar el modelo que mejor represente al objeto.

El sistema se compara contra el mismo pero en lugar de usar AGC utiliza métodos RANSAC de la biblioteca Point Cloud Library (PCL). Los resultados muestran valores similares entre los sistemas así como algunas ventajas individuales para cada sistema.

ABSTRACT

It is presented the development of an interface between real objects and virtual reality, with the help of an object classifier the objects are represented by simple geometric forms. By modifying the method using conformal geometric algebra (CGA) is expected to improve the classifier.

A parameter estimation is made for the sphere, plane, and cylinder using the RANSAC estimator and the Kinect to get the point clouds.

The system is compared with a modified version of itself which use the RANSAC methods from the Point Cloud Library (PCL) instead of the CGS methods, the results show some similarity between systems and some benefits for each system.

CONTENIDO

Resumen	iii
Abstract	v
Contenido	vii
Índice de Figuras	ix
Índice de Tablas	xi
Índice de algoritmos	xv
1 Introducción	1
1.1 Marco teórico	1
1.2 Estado del Arte	11
1.3 Planteamiento del Problema	15
1.4 Justificación.	15
1.5 Objetivo General	16
1.6 Objetivos Particulares	16
1.7 Solución Propuesta	16
1.8 Aportaciones	16
2 Desarrollo	19
2.1 Metodología	19
2.2 Pre-Procesamiento.	20
2.3 Clasificación de objetos.	24
2.4 Descripción General del Sistema.	29
2.5 Álgebra Geométrica Conforme	30
3 Pruebas y Caso de estudio	35
3.1 Prueba de Captura de Datos	35
3.2 Prueba de Segmentación	36
3.3 Descripción del Caso de Estudio	37
4 Análisis de Resultados.	43
4.1 Objeto Esférico	43
4.2 Objeto Plano	45
4.3 Objeto Cilíndrico	45
4.4 Rendimiento	46
5 Consideraciones Finales	47
5.1 Conclusiones	47
5.2 Trabajo a futuro	47
Referencias	49
Apéndices	51
A Pruebas	55
B Códigos	65

ÍNDICE DE FIGURAS

1.1	Control WiiMote de Nintendo	3
1.2	Control PS/Move de Sony	3
1.3	Sensor Kinect 1	4
1.4	Sensor Kinect 2.0	5
1.5	Diagrama de flujo del método RANSAC	7
1.6	Calculo de kinematica inversa para un robot simple	12
1.7	Sistema de un robot medico.	12
1.8	Segmentación de una imagen.	13
1.9	Técnica de pre-procesamiento	14
1.10	Reconocimiento de objetos usando normales.	14
1.11	Modelado de un cilindro usando AGC.	15
1.12	Diagrama del sistema propuesto	17
2.1	Diagrama a bloques del sistema	19
2.2	Logo de la Biblioteca de Nube de Puntos	20
2.3	Datos obtenidos del dispositivo Kinect.	21
2.4	Filtro de densidad de puntos.	22
2.5	Operación resta.	23
2.6	Prueba de la transformada de Hough	24
2.7	Diagrama de flujo del sistema	30
2.8	Distancia entre un punto y una esfera.	31
2.9	Programa GAViewer	34
3.1	Prueba para el filtro de densidad de puntos.	36
3.2	Estudio de pruebas	38
3.3	Prueba para la esfera	39
3.4	Prueba para el plano	41
3.5	Prueba para el cilindro	42
4.1	Tiempos promedios de la prueba	46

ÍNDICE DE TABLAS

1.1	Tabla comparativa de los modelos del Kinect	4
1.2	Entidades geométricas del AGC en su forma IPNS y OPNS	11
2.1	Valores calculados de E(k)	28
3.1	Descripción de los tiempos del sistema	41
4.1	Resumen de tiempos de la prueba para la esfera usando AE	46

ÍNDICE DE CÓDIGOS

4.1	Lectura de datos en R	44
4.2	Asignación de rangos.	44
4.3	Calculo de z.	44
A.1	Datos obtenidos para la prueba del plano usando AE.	55
A.2	Datos obtenidos para la prueba del plano usando AGC.	57
A.3	Datos obtenidos para la prueba de la esfera usando AE.	59
A.4	Datos obtenidos para la prueba de la esfera usando AGC.	60
A.5	Datos obtenidos para la prueba del cilindro usando AE.	62
A.6	Datos obtenidos para la prueba del cilindro usando AGC.	64
B.7	Archivo test.cpp	66
B.8	Archivo ransac.h	78
B.9	Archivo objects.h	89

ÍNDICE DE ALGORITMOS

2.1	Agrupamiento	23
2.2	RANSAC	26
2.3	RANSAC modificado	29

GLOSARIO

algoritmo	Conjunto de instrucciones ordenadas y finitas para realizar alguna actividad 7, 23, 25, 29
cilindro	Elemento geométrico descrito por la revolución de un rectángulo. 16, 27, 32, 38, 40, 45
conforme	Hace referencia a las transformaciones conformes, en los cuales se mantienen los ángulos. 9, 10
controlador	Programa que permite al sistema operativo interaccionar con un periférico 21
esfera	elemento geométrico descrito por una superficie curva cuyos puntos equidistan al centro de esta. 16, 26, 31, 38
filtro	Programa que remueve parte de la información dada. 22
imagen	Representación visual de algún objeto 2
interfaz	Conexión funcional entre dos sistemas, programas, dispositivos o componentes de cualquier tipo 1
método	Subrutina perteneciente a la clase de algún programa para realizar una acción 6
nube de puntos	Conjunto de puntos, habitualmente describen superficies. 2, 20
pixel	Del inglés "picture element" es la unidad básica de una imagen 2
plano	Elemento geométrico de dos dimensiones, que contiene infinitos puntos y rectas. 16, 26, 32, 38, 40, 45
punto	Es una figura geométrica que define una ubicación específica del espacio. 2
realidad virtual	Entorno de escenas u objetos de apariencia real, generado mediante tecnología informática 1
sensor	Dispositivo eléctrico con la capacidad de manifestar magnitudes en variables eléctricas 20
sistema	Objeto complejo cuyos componentes se relacionan con al menos algún otro componente. 16, 29

1

INTRODUCCIÓN

Actualmente los sistemas de **realidad virtual** se encuentran en auge tanto en el ámbito comercial como en el desarrollo tecnológico, esto gracias al incremento en la capacidad de cómputo y la reducción del tamaño de algunos dispositivos electrónicos. Tecnologías como Google Cardboard, HTV VR Headset, Microsoft Hololens, etc. han sido tecnologías pioneras permitiendo al usuario experimentar la realidad virtual de forma más personal e intuitiva.

De manera similar a cuando la tecnología de las pantallas táctiles llegó a los celulares y forzaron la creación de nuevas interfaces para la interacción entre el usuario y la computadora, las nuevas tecnologías de realidad virtual se enfrentan al desarrollo de nuevas **interfaces**.

Estas nuevas interfaces son particularmente complicadas ya que se busca una interacción intuitiva para el usuario, y que no solo se busca la interacción entre usuario y el mundo virtual, algunas empresas apuntan a la intersección entre la realidad virtual y la realidad aumentada (realidad mixta), esto conlleva una interacción entre elementos reales, elementos virtuales y el usuario o usuarios.

El problema de interacción entre el usuario y el mundo virtual se ha intentado trabajar usando sistemas de captura de movimiento como los de la empresa VICON [1], usando cámaras infrarrojas capaces de medir la distancia entre la cámara y un reflector de luz infrarroja es posible calcular la posición y orientación de un objeto. El problema es que estos sistemas son delicados y costosos.

La búsqueda de diferentes formas de realizar la interacción entre un entorno virtual, el mundo real y el usuario representa un reto importante.

1.1 MARCO TEÓRICO

Para el desarrollo de este trabajo se abordan temas como la obtención de datos (visión por computadora), clasificación (RANSAC) y la implementación de AGC. Por lo cual, a continuación se describen los conceptos necesarios para el desarrollo.

1.1.1 Representación Digital

Una imagen digital es guardada según el formato de imagen (TIFF, BMP, JPEG, etc), pero todas las imágenes luego de su interpretación son representadas en una computadora por un arreglo bidimensional de **píxeles**.

El arreglo de píxeles se puede ver como un conjunto de matrices numéricas, dependiendo la representación del píxel (RGB, CMY, RGBA, CMYA, etc.) son los canales necesarios para representarlo por completo. Así una imagen en RGB (rojo, verde, azul por sus siglas en inglés) cuenta con tres canales, cada canal define la intensidad de cada color, así para mostrar una imagen $M \times N$ en RGB es necesario tres matrices de M filas y N columnas.

De esta manera es posible generar una imagen nueva a partir de otras realizando operaciones sobre imágenes como la suma, resta y multiplicación, aplicando una transformación como se muestra en la ecuación (ec. 1).

$$q(x, y) = f(p_1(x, y), p_2(x, y)) \quad (1)$$

Donde p_1 es un píxel de la imagen 1 y p_2 un píxel de la imagen 2 y $q(x, y)$ el píxel resultante de la nueva imagen [2].

Una **nube de puntos** se describe como el conjunto de puntos en un espacio dado o una estructura de datos usada para representar una colección de puntos [3]. Al trabajar con nubes de puntos se pueden pensar en ellas como una imagen de tres dimensiones y en lugar de un píxeles se trabajan sobre **puntos**. Al igual que las imágenes.

A diferencia de una **imagen** una nube de puntos puede o no estar organizada en filas y columnas, sensores como el Kinect en los que se obtiene una imagen de profundidad se obtiene una nube de puntos de $M \times N$ puntos, mientras que otras nubes de puntos tienden a ser solo una lista $M \times 1$ de puntos en el espacio [4].

1.1.2 Sensor Kinect

En el desarrollo de video juegos, la competencia entre grandes empresas ha dado como resultado en una variedad de dispositivos que le permiten al usuario una interacción más natural con entornos virtuales. Estos dispositivos son variados tanto en sus componentes físicos como en la forma de interacción. unos de los dispositivos mas conocidos son el WiiMote, PS/Move y el Kinect.

Para la consola Wii de Nintendo se usa un WiiMote, mostrado en la figura 1.1, que con el uso de una barra de leds infrarrojos y sensores de posición y aceleración,

es capaz de obtener su posición respecto la barra de leds [5].



Figura 1.1: Control WiiMote de Nintendo

En el caso del PlayStation de Sony, que usa un PS/Move, mostrado en la figura 1.2, se usa una cámara de vídeo que junto con sensores de posición y aceleración, se realiza esta interacción.



Figura 1.2: Control PS/Move de Sony

Mientras que Xbox de Microsoft optó por un dispositivo realizado por la empresa PrimeSense llamado Kinect, el cual integra micrófonos, un motor, un emisor infrarrojo, una cámara de vídeo a color y otra infrarroja [5], el dispositivo se muestra en la figura 1.3 [6]. Usando la cámara infrarroja el Kinect detecta la profundidad de objetos usando una técnica llamada tiempo de vuelo (Time of Flight ToF), en la cual se obtiene el tiempo que tarda un rayo de luz en rebotar en un objeto, sabiendo la velocidad del rayo de luz y el tiempo desde su emisión hasta que es captado por la cámara, se calcula la distancia del objeto al sensor[7].

El dispositivo Kinect cuenta con dos versiones el primero se anuncio con la consola Xbox 360 en el 2010 y el Kinect 2.0, mostrado en la figura 1.4, se anuncio junto a la Xbox One en 2013 los cuales cuentan con las características mostradas en la tabla 1.1



Figura 1.3: Sensor Kinect 1

[5].

Tabla 1.1: Tabla comparativa de los modelos del Kinect

Característica	Kinect 1	Kinect 2
Cámara de color	640 x 480 @30 fps	1920 x 1080 @30 fps
Cámara de profundidad	320 x 240	512 x 424
Profundidad Maxima	~4.5 m	~4.5 m
Distancia de profundidad minima	40 cm	50 cm
Visión de campo horisontal	57 grados	70 grados
Visión de campo vertical	43 grados	60 grados
Motor de inclinación	si	no
Uniones definidas para esqueletos	20 uniones	26 uniones
Esqueletos rastreados	2	6
Estandar USB	2.0	3.0

1.1.3 Visión Por Computadora

La visión por computadora trata de describir el mundo que vemos a partir de una o varias imágenes y reconstruir sus propiedades, como forma, color e intensidad de luz, tratando de imitar la visión de una persona. Es usada en gran cantidad de aplicaciones como: reconocimiento óptico de letras, inspección de productos en fábricas, modelado 3D, reconocimiento facial, etc. [8].

Se llevan a cabo varios procesos antes de someter una imagen a reconocimiento, esto sirve para resaltar puntos importantes, facilitar la detección de características, eliminar ruido, etc. Estos procesos son diferentes según el problema a tratar, pero



Figura 1.4: Sensor Kinect 2.0

ayudan en gran medida mejorar el reconocimiento de objetos por la computadora, y en conjunto se les conoce como pre-procesamiento de imagen.

Para que la computadora sea capaz de reconocer su entorno existen diferentes técnicas como la selección de puntos característicos o de marcadores en un objeto visto por dos o más cámaras y el cálculo de su posición usando las imágenes obtenidas; también el análisis de la perspectiva de una imagen (buscando rectas o círculos) o algoritmos de búsqueda como los usados en sistemas con inteligencia artificial (redes neuronales o K vecinos más próximos).

Transformada de Hough

El algoritmo usado para la transformada de Hough (TH) fue patentado en 1962 por P. Hough como un método para reconocer líneas rectas y más tarde se extendió para detectar figuras complejas, como círculos, polígonos o elipses, y se le llamó la transformada de Hough generalizada, esta técnica permite la detección de los parámetros de un modelo usando un sistema de votación en el cual cada dato favorece al conjunto de parámetros que satisfacen al modelo para dicho dato, aunque la TH se ha visto que tiene dificultad para trabajar con modelos con gran cantidad de incógnitas[9].

Por ejemplo si quisieramos detectar una línea recta, lo primero que se necesita es saber la definición de una línea recta, esta está definida como la sucesión de puntos que se extienden en una misma dirección. y su representación matemática está dada por la ecuación (ec. 2)

$$y = mx + b \quad (2)$$

Donde x y y son las variables, m la pendiente de la linea y b la intersección de la linea sobre el eje y respecto del origen. En este ejemplo m y b son los parámetros de una linea recta, los cuales describen la linea que esta buscando.

Luego de conocer el modelo de la linea que se esta buscando se necesitan los posibles candidatos, es decir de una imagen cuales son los posibles puntos que pertenecen a una linea, una forma de obtenerlos es el resaltar los bordes de una imagen, recordando que un borde se entiende como un cambio súbito de un valor, para esto se pueden realizar varios filtros, como el filtro de Sobel, Roberts, Prewitt, etc. , de los cuales se obtiene una imagen binaria, con un valor de uno indicando la presencia de un borde y cero en el caso contrario.

Ya obtenidos los posibles puntos, lo siguiente es aplicar la TH a cada punto, lo que realiza la TH es describir el modelos usando los parámetros, en lugar de tener los ejes (x, y) se usan los ejes (m, b) , de esta forma un punto en (x, y) representa una linea en el espacio de parámetros (m, b) , y una linea en (x, y) se representa como un punto. Estas consideraciones son importantes ya que encontrando la intersección de lineas en el espacio de parámetros (m, b) se consigue una linea en (x, y) , ya que no se conoce la intersección de estas lineas pueden estar en el infinito, se utilizan las coordenadas polares de esta manera las lineas se forman por funciones periódicas y le es más fácil encontrar estas intersecciones a la computadora, la representación de la linea en coordenadas polares queda descrita como en la ecuación (ec. 3), donde p es la distancia qu e existe del origen al punto (x, y) , y θ es el angulo del vector perpendicular a la recta y que pasa por el origen.

$$y = \left(-\frac{\cos\theta}{\sin\theta} \right) * x + \left(\frac{p}{\sin\theta} \right) \quad (3)$$

Escribiendo la ecuación en función de θ y despejando p se obtiene la ecuación (ec. 4), que para cada punto (x, y) se obtiene una sinusoida en el espacio de parámetros (p, θ) donde $\theta \in [0, \pi]$ y $p \in \mathbb{R}$ o $\theta \in [0, 2\pi)$ y $p \geq 0$, de esta manera un punto vota por toda una familia de rectas, cada que una sinusoida interseca con otra votan por la recta que corresponde al punto p, θ), y los mas votados son los que mas probabilidad tienen de ser una de las lineas que se buscan.

$$p = x * \cos\theta + y * \sin\theta \quad (4)$$

1.1.4 RANSAC

Del inglés RANdom SAmple Consensus (RANSAC) es un [método](#) iterativo para estimar parámetros de algún modelo matemático, es decir, encuentra el conjunto de parámetros para un modelo matemático que mejor describe a un conjunto de datos [\[10\]](#).

Este método es no determinista ya que realiza el muestreo de forma aleatoria y dependiendo de la distribución de los datos es la precisión que se puede obtener.

Entre mas repeticiones se realicen mayor precisión se tendrá ya que se habrán usado un mayor número de combinaciones.

En la figura 1.5 se muestra el diagrama de flujo que se sigue para implementar el [algoritmo RANSAC](#).

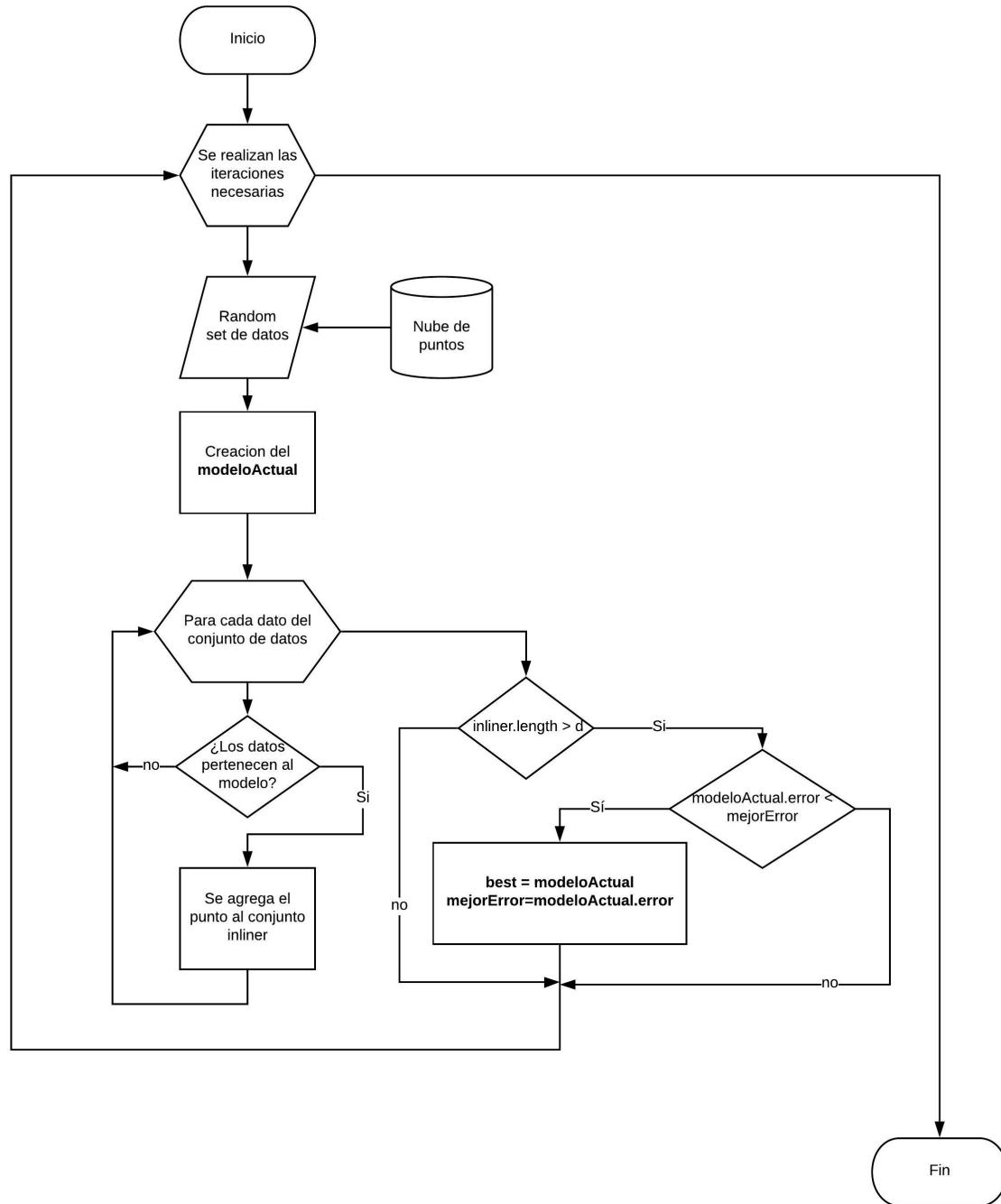


Figura 1.5: Diagrama de flujo del método RANSAC

1.1.5 Álgebra Geométrica (AG)

En matemáticas es común el uso de álgebras de Clifford y Grassmann, aunque por cuestiones prácticas para la ingeniería se prefiere trabajar con el álgebra vectorial para espacios euclidianos (AE) propuesta por Josiah Willard Gibbs en el siglo XIX, la cual se enfoca a la manipulación de tres dimensiones y los números escalares, aunque últimamente el uso de AG a sido necesaria por la demanda de los físicos para trabajar en mas dimensiones [11].

Al hablar de álgebras nos referimos a un con junto de operaciones con estructura algebraica. Para tener una estructura algebraica de deben cumplir las siguientes reglas. Dado el conjunto $A(S)$ de todas las aplicaciones inyectivas de S sobre sí mismo, siendo $A(s)$ un conjunto no vacío [12]:

1. Cerradura, a saber, si $f, g \in A(S)$, entonces $fg \in A(S)$. Se dice que $A(S)$ es cerrado respecto a este producto.
2. Asociatividad, esto es, dadas $f, g, h \in A(S)$, entonces $f(gh) = (fg)h$.
3. Existencia de un elemento unidad, a saber, existe un elemento particular $i \in A(S)$ (la aplicación identidad) tal que $fi = if = f$ para toda $f \in A(S)$.
4. Existencia de inversas, esto es, dada $f \in A(S)$ existe un elemento en $A(S)$, denotado por f^{-1} , tal que $ff^{-1} = f^{-1}f = i$.

El álgebra vectorial V esta definido por el espacio vectorial conformado por el conjunto de objetos denominados vectores, junto con dos operaciones binarias llamadas suma y multiplicación por un escalar que satisface los axiomas [13]:

1. Si $x \in V$ y $y \in V$, entonces $x + y \in V$ (cerradura bajo suma)
2. Para todo $x, y, z \in V$, $(x + y) + z = x + (y + z)$ (ley asociativa de la suma de vectores)
3. Existe un vector $0 \in V$ tal que para todo $x \in V$, $x + 0 = 0 + x = x$ (el 0 se llama vector cero o idéntico aditivo)
4. Si $x \in V$, existe un vector $-x \in V$ tal que $x + (-x) = 0$ ($-x$ se llama inverso aditivo de x)
5. Si x y $y \in V$, entonces $x + y = y + x$ (ley conmutativa de suma de vectores)
6. Si $x \in V$ y α es un escalar, entonces $\alpha x \in V$ (cerradura bajo multiplicación por un escalar)
7. Si x y $y \in V$ y α es un escalar, entonces $\alpha(x + y) = \alpha x + \alpha y$ (primera ley distributiva)

8. Si $x \in V$ y α y β son escalares, entonces $(\alpha + \beta)x = \alpha x + \beta x$ (segunda ley distributiva)
9. Si $x \in V$ y α y β son escalares, entonces $\alpha(\beta x) = (\alpha\beta)x$ (ley asociativa de la multiplicación por escalares)
10. Para cada vector $x \in V$, $1x = x$

El AG es un marco de trabajo matemático que permite desarrollar algoritmos de forma rápida e intuitiva. Las ventajas del uso del AG radican en la unificación de varios sistemas matemáticos (álgebra vectorial, números complejos, cuaterniones, etc.), así como el manejo intuitivo de objetos y operaciones geométricas[11].

El álgebra abstracta desempeña una función: la de vínculo unificador entre ramas dispares de la matemática y la de área de investigación.

El desarrollo del AG se remonta a 300 B.C. con Elucides y la geometría sintética u ha ido evolucionando hasta el presente. La geometría analítica de Descartes (1637), el álgebra compleja de Wessel y Gauss (1798), el álgebra de Hamilton (1843), el álgebra exterior de Grassmann (1844), el álgebra matricial de Cayley (1854), el álgebra de Clifford (1878), el álgebra tensorial de Ricci (1890), las formas diferenciales de Cartan (1923), y el álgebra de giros (spin) de Pauli y Dirac (1928) contribuyeron al desarrollo del AG. El AG representa geometrías como entidades y a estas se les denominan multivectores, también se presenta el producto geométrico como la operación para el cálculo de multivectores[14]

A mitad del siglo XIX J. Liuville probó para el caso de tres dimensiones que cualquier transformación conforme en todo \mathbb{R}^n puede expresarse como una composición de inversiones en esferas (inversions in spheres) y reflexiones en hiperplanos (reflections in hyperplanes). En particular la transformación de rotación, traslación, dilatación e inversión se obtienen de las transformaciones anteriores. En el Álgebra Geométrica Conforme (AGC) estos conceptos se simplifican por el isomorfismo entre el grupo **conforme** en \mathbb{R}^n y el grupo de Lorentz en \mathbb{R}^{n+1} , se puede expresar con una transformación lineal de Lorentz, una transformación conforme no lineal, y la representación usando versores para simplificar la composición de transformaciones con la multiplicación de versores, lo cual, a diferencia del álgebra matricial, resulta en una forma simple y eficiente de interpretar la geometría de transformaciones conformes[14].

Las álgebras de Clifford o Álgebra Geométrica(AG) creadas y clasificadas por William Kingdon Clifford. Incorpora una regla de multiplicación para vectores usados en el álgebra exterior de Hermann Grassmann ($\Lambda\mathbb{R}$). En el caso especial de los cuaterniones de Hamilton se incorpora al álgebra de Clifford.

Un AG de Clifford $Cl(\mathbb{R}^n)$ representa un álgebra real asociativa de n dimensiones. Se define esta álgebra usando un conjunto de vectores base $\{e_1, \dots, e_n\}$, donde $e_i^2 = 1$,

$i = 1, 2, \dots, n$, y los elementos e_i son anticonmutativos entre ellos. Donde el álgebra comprende a todos los escalares (Λ^0), vectores e_i (Λ^1), bivectores $e_i e_j$ (Λ^2), trivectores $e_i e_j e_k$ (Λ^3), etc. formados por los vectores base de orden 0 a n , respectivamente.

Un vector $v = v_1 e_1 + v_2 e_2 + \dots + v_n e_n$, donde v_1, v_2, \dots, v_n son escalares que cumplen con las reglas del álgebra y asumimos la asociatividad de la multiplicación antes que la de la suma, obtenemos $v^2 = v_1^2 + v_2^2 + \dots + v_n^2$.

Para dos vectores se define el producto geométrico como:

$$vw = \frac{1}{2}(vw + wv) + \frac{1}{2}(vw - wv) = v \cdot w + v \wedge w \quad (5)$$

Donde podemos ver que el producto geométrico (ec. 5) se define como la suma del producto punto (ec. 6) y el producto externo (ec. 7). El producto externo es anticonmutativo $v \wedge w = -w \wedge v$ y es una combinación de bivectores $e_i e_j$. En el caso de tres dimensiones el producto externo corresponde con el producto cruz ($a \times b$).

$$v \cdot w = \frac{1}{2}(vw + wv) \quad (6)$$

$$v \wedge w = \frac{1}{2}(vw - wv) \quad (7)$$

Se define al multivector r -vector como el producto externo de r vectores independientes. Normalmente el producto externo $v \wedge w$ entre dos vectores define un plano orientado en n dimensiones donde v y w pertenecen a este plano.

Álgebra Geométrica Conforme (AGC)

El álgebra geométrica **conforme** es la combinación del AG de Clifford y una geometría euclíadiana hiperbólica. Nikolai Lobachevsky en el siglo XIX obtiene que para geometrías no-Euclidianas, en espacios con estructuras hiperbólicas existen subconjuntos isomorfos al espacio Euclidiano, para esto Lobachevsky enuncia dos restricciones para $x_c \in \mathbb{R}^{n+1,1}$, la primera restricción es la representación homogénea obtenida, normalizando el vector x_c tal que $x_c \cdot e_\infty = -1$, y la segunda restricción es que el vector x_c debe ser un vector nulo, $x_c^2 = 0$ [14].

El AGC en el campo de Visión por Computadora es usada en la geometría proyectiva que se enfoca en cómo las imágenes son proyectadas de un mundo tridimensional a la pantalla de una cámara o una retina; por su enfoque de proyección, el AGC permite la linealización de trasformaciones que de otra forma serían no lineales y las relaciones de incidencia entre puntos, líneas y planos son expresados de una forma eficiente [15].

AGC usa los versores Euclidianos (e_1, e_2, e_3), mas e_+, e_- , los símbolos + y - se refieren al signo que se obtiene al sacar su cuadrado, al igual que los vectores base Euclidianos $e_+^2 = 1$ y a diferencia de los anteriores $e_-^2 = -1$ y el producto interno entre ellos da cero $e_+ \cdot e_- = 0$. Estos cinco versores son los que extienden el espacio $\mathbb{R}^{3+1,1}$, donde los primeros cuatro versores cumplen con $e_i^2 = 1$ y el restante $e_-^2 = -1$. Estos versores se pueden escribir como se muestra en (ec. 8)

$$e_0 = \frac{1}{2}(e_- - e_+), e_\infty = e_- + e_+ \quad (8)$$

Escrito de esta manera e_0 representa el origen en 3D, y e_∞ representa el infinito. Estas representaciones ayudan a visualizar los elementos de forma intuitiva.

Las propiedades estos nuevos vectores son diferentes a e_+ y e_- algunas diferencias son que sus cuadrados dan cero $e_0^2 = e_\infty^2 = 0$ y el producto interno $e_\infty \cdot e_0 = -1$

En la tabla 1.2 se observan las entidades geométricas y dos representaciones algebraicas IPNS y OPNS, por sus iniciales en inglés Inner Product Null Space y Outer Product Null Space respectivamente. estas representaciones tienen la cualidad de ser duales. Un elemento dual es aquel que se obtiene al hacer el producto interno con el inverso del pseudo-escalar $I = e_1 \wedge e_2 \wedge e_3 \wedge e_\infty \wedge e_0$. El dual de X esta dado por $X^* = X \cdot I^{-1}$

Tabla 1.2: Entidades geométricas del AGC en su forma IPNS y OPNS

Entidad	IPNS	OPNS
Punto	$P = x + \frac{1}{2}x^2 e_\infty + e_0$	
Esfera	$S = P - \frac{1}{2}r^2 e_\infty$	$S^* = P_1 \wedge P_2 \wedge P_3 \wedge P_4$
Plano	$\pi = n + d e_\infty$	$\pi^* = P_1 \wedge P_2 \wedge P_3 \wedge P_\infty$
Círculo	$Z = S_1 \wedge S_2$	$Z^* = P_1 \wedge P_2 \wedge P_3 \wedge P_4$
Línea	$L = \pi_1 \wedge \pi_2$	$L^* = P_1 \wedge P_2 \wedge e_\infty$
Par de puntos	$Pp = S_1 \wedge S_2 \wedge S_3$	$Pp = P_1 \wedge P_2$

Un ejemplo del uso del AGC se presenta en el libro titulado Foundations of Geometric Algebra Computing [11], es el calculo de la kinematica inversa para un robot simple mostrado en la figura 1.6 , donde se buscan los angulos necesarios para posicionar el robot en una posición dada. usando esferas y planos para representar los movimientos del robot, se buscan las intersecciones entre estas figuras para encontrar la posición y posteriormente los ángulos necesarios para posicionar el robot.

1.2 ESTADO DEL ARTE

En esta sección se describen algunos trabajos que se han realizado. Algunos para buscar mejorar la visión por computadora, otros haciendo más robustos los algorit-

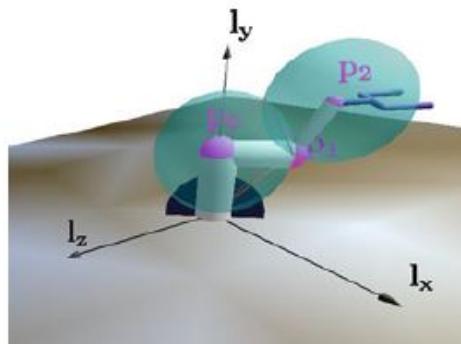


Figura 1.6: Calculo de kinematica inversa para un robot simple

mos y otros usando alguna alternativa a lo ya obtenido. Cada trabajo aporta nuevo conocimiento que ayudan con el desarrollo de este trabajo.

1.2.1 Visión para Robot Médico usando Álgebra Geométrica Conforme

En el artículo presentado con el título de “Medical Robot Vision using the Conformal Geometric Algebra Framework” [16], se propone una forma de manejar la calibración mano-ojo de la visión para un robot médico diseñado para cirugías mostrado en la figura 1.7a, donde se adecuó la teoría del tornillo usando AGC, la arquitectura del sistema se muestra en la figura 1.7b, se usa el sensor Kinect para generar el modelado 3D y una interfaz háptica para mejorar el manejo del robot.

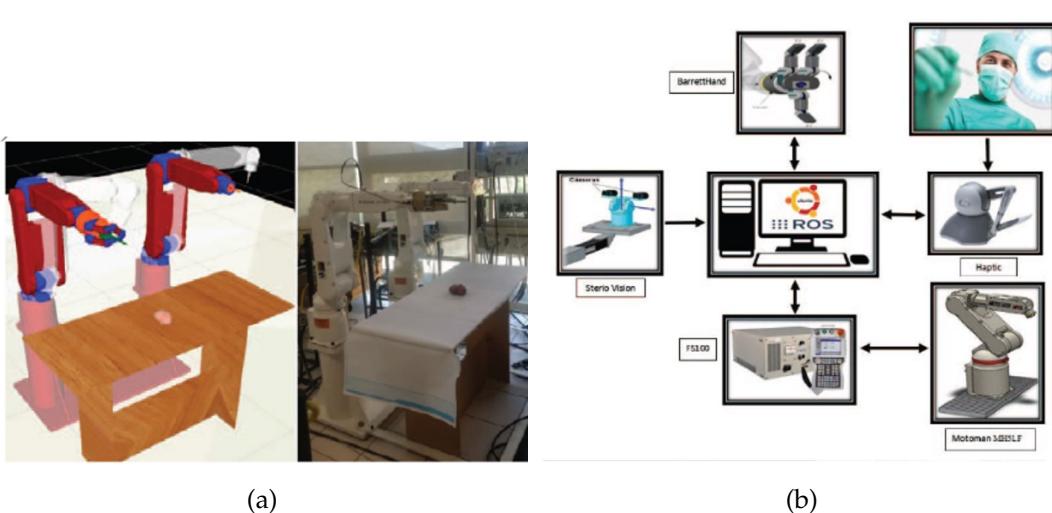


Figura 1.7: Sistema de un robot medico, (a) Imágenes del sistema de brazos robot, (b) Diagrama del sistema médico robot.

1.2.2 Visión por Computadora Mejorada con el Sensor Kinect de Microsoft

En el artículo con título “Enhanced Computer Vision with Microsoft Kinect Sensor: A Review” [17], se describen algunas técnicas de Visión por Computadora con imágenes RGB que fueron modificadas o mejoradas usando el sensor Kinect como la que se muestra en la figura 1.8, se describe una introducción al uso del Kinect en temas como el pre-procesamiento (usando la profundidad como filtro como se muestra en la figura 1.8c), el seguimiento y reconocimiento de objetos (que permite dar su rotación y posición) y el reconocimiento de objetos en una escena (solo predice si existe un objeto).

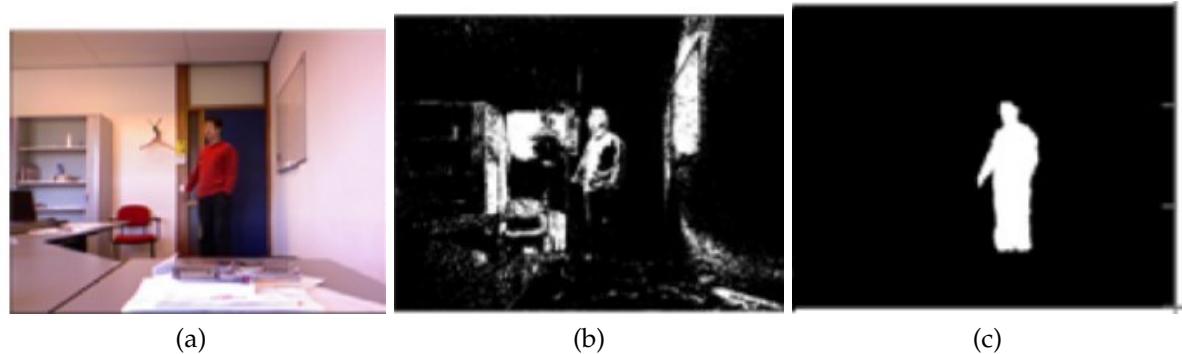


Figura 1.8: Segmentación de una imagen, (a) Imagen RGB, (b) Máscara FG usando RGB, (c) Máscara FG usando profundidad.

1.2.3 Mejorando la Clasificación y Detección de Objetos Basado en la Transformada de Hough y el Sensor Kinect

La transformada de Hough se usa en el procesamiento de imágenes ya que permite detectar formas simples como rectas, circunferencias o elipses. En el artículo titulado “Improving 3D Object Detection and Classification Based on Kinect Sensor and Hough Transform” [9], se propone un algoritmo que usa puntos característicos y el espectro del color como dos procesos intercalados que cooperativamente reconocen objetos al estilo 2.5D, para automatizar el pre-procesamiento de una escena, en la figura 1.9 se observan las etapas del algoritmo.

1.2.4 Reconocimiento de Objetos Mediante Sensor 3D Kinect

En el proyecto titulado “Reconocimiento de Objetos Mediante Sensor 3D Kinect” [18], se utiliza un sensor Kinect 3D para obtener una malla de puntos y reconocer un grupo de objetos usando un clasificador basado en vectores. Obteniendo una nube de puntos del sensor 3D Kinect calculan los vectores normales al plano en cada punto del objeto, así como se muestra en la figura 1.10, y se determina qué tipo de

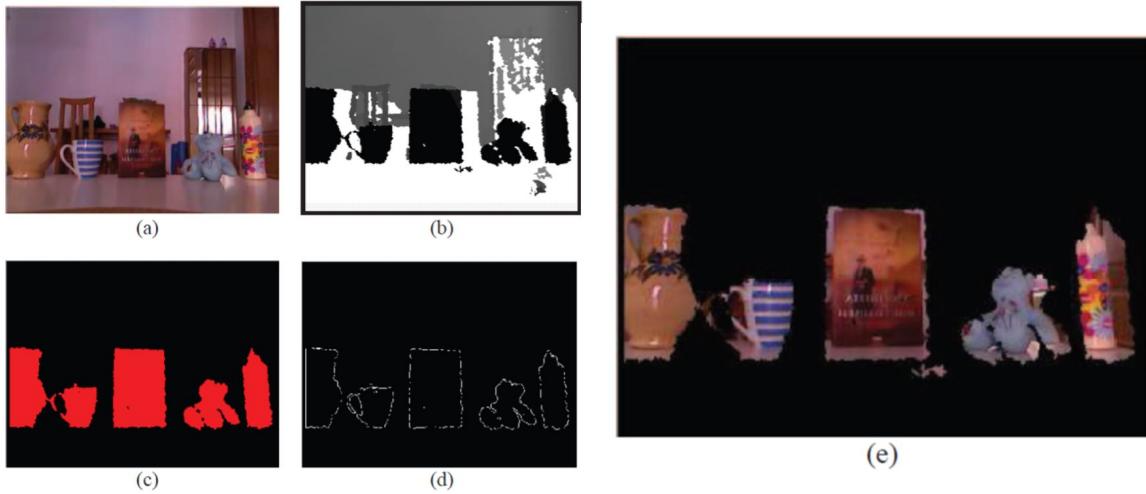


Figura 1.9: Técnica de pre-procesamiento, (a) Imagen RGB, (b) Imagen de profundidad, (c) Imagen binaria, (d) Filtro Canny, y (e) Segmentación

objeto es si cumple con las características dadas para cada figura.

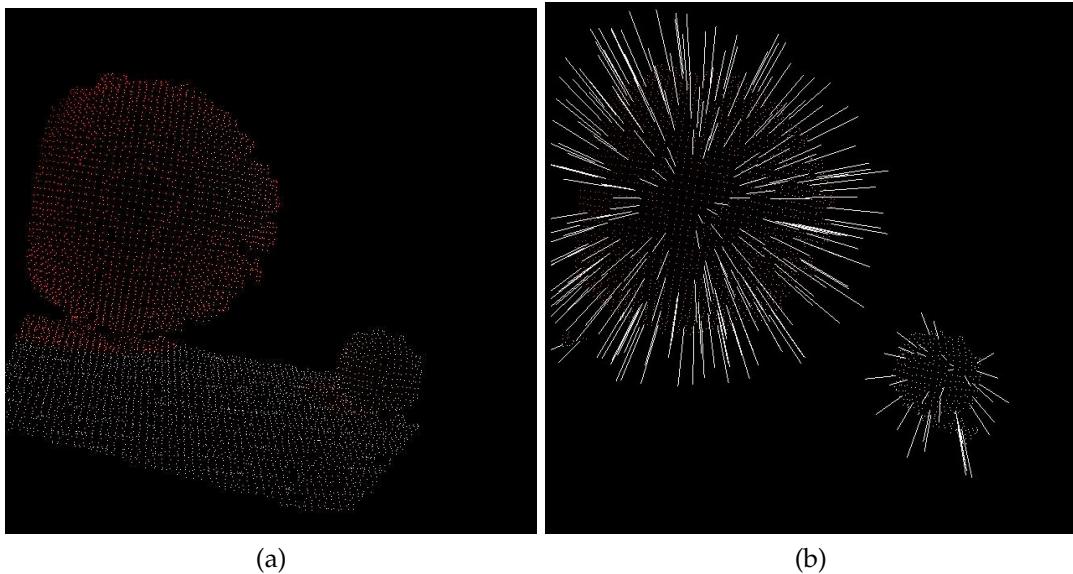


Figura 1.10: Reconocimiento de objetos usando normales, (a)Nube de puntos obtenidos del sensor 3D Kinect, (b) Visualización de las normales calculadas.

1.2.5 Detección de Objetos en Nubes de Puntos Usando Álgebra Geométrica Conforme

En el trabajo “Object Detection in Point clouds Using Conformal Geometric Algebra” [19], se usa el algoritmo RANSAC para detectar objetos con formas geométricas, así como también proponen varias formas de calcular cilindros usando AGC, primero usando una circunferencia extrudía sobre la normal del plano que pasa por el círculo

y otra usando dos esferas como se muestra en la figura 1.11.

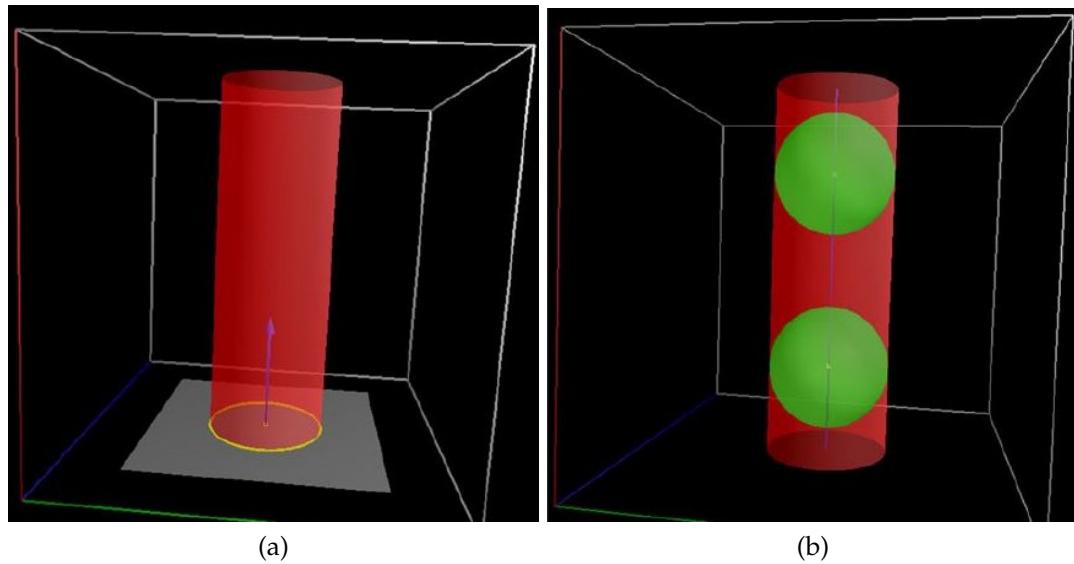


Figura 1.11: Modelado de un cilindro usando AGC, (a) Cilindro construido usando circulo y su normal, (b) Cilindro construido con dos esferas y la linea que une sus centros.

1.3 PLANTEAMIENTO DEL PROBLEMA

En la actualidad existen diversos sistemas que permiten la interacción y el reconocimiento de objetos del mundo real, pero la mayoría de ellos se limitan a trabajar solo con objetos o entornos ya establecidos. Sistemas de realidad virtual como HTC Vive [20] y Oculus Rift [21] interactúan con objetos virtuales usando un mando especial, pero no detectan objetos con los cuales el usuario pudiera colisionar. En sistemas de realidad mixta como los Holo-Lens de Microsoft [22] o en visión por computadora se puede obtener la posición de las paredes de una habitación y encontrar obstáculos, pero la interacción del usuario es limitada. Por lo anterior, se hace clara la falta de proyectos que permitan la interacción con objetos cotidianos sin la necesidad de conocer a detalle todas las características del objeto, solo interesando algunas de ellas, tales como su tamaño, posición y geometría.

1.4 JUSTIFICACIÓN

Con el desarrollo de este proyecto se obtendrán las bases para una interfaz de realidad mixta, que beneficiará a los desarrolladores de experiencias virtuales. Esto debido a que se presenta un sistema para representar objetos reales en un mundo virtual, lo cual constituye una primera aproximación al desarrollo de una interfaz entre usuario, maquina y entorno; el uso del álgebra geométrica conforme permite describir geometrías de forma mas intuitiva, lo que puede facilitar e incluso mejorar dicha representación.

1.5 OBJETIVO GENERAL

El objetivo general de este trabajo es diseñar, desarrollar e implementar un sistema que represente objetos reales como geometrías básicas ([esferas](#), [planos](#) o [cilindros](#)) en un espacio virtual, usando el sensor Kinect y un algoritmo desarrollado con AGC.

1.6 OBJETIVOS PARTICULARES

Los objetivos particulares derivados del objetivo general son los siguientes:

- Investigar las herramientas para el uso del sensor Kinect.
- Desarrollar el módulo encargado de la captura de datos del sensor Kinect.
- Desarrollar el módulo encargado de la segmentación de los datos obtenidos.
- Investigar y desarrollar un algoritmo de reconocimiento de objetos usando los datos del Kinect.
- Desarrollar un ambiente virtual para mostrar los resultados del algoritmo.
- Investigar las herramientas que usen AGC.
- Modificar el algoritmo de reconocimiento de objetos para que trabaje con AGC.
- Realizar pruebas de rendimiento para el algoritmo de reconocimiento.

1.7 SOLUCIÓN PROPUESTA

En la figura 1.12 se observa un diagrama del [sistema](#) propuesto donde se destacan las interacciones del sistema. Se inicia con un objeto del mundo real que es observado por el sensor Kinect, el cual obtiene imagen RGB, infrarroja y las distancias de una matriz de puntos, que pasan por un pre-procesamiento para eliminar datos no deseados; usando un algoritmo de reconocimiento se obtiene la aproximación del objeto a una figura simple la cual es mostrada en un ambiente virtual creado en la computadora.

1.8 APORTACIONES

En este proyecto se tiene como aporte el desarrollo de una representación de objetos para una interfaz de realidad mixta, en la cual se busca una representación de objetos sin preocuparse de que objeto es, este tipo de interfaces no necesitan conocer a detalle los objetos, con una descripción general de su geometría y su posición junto a la integración de un ambiente virtual permite el desarrollo de sistemas en realidad mixta.

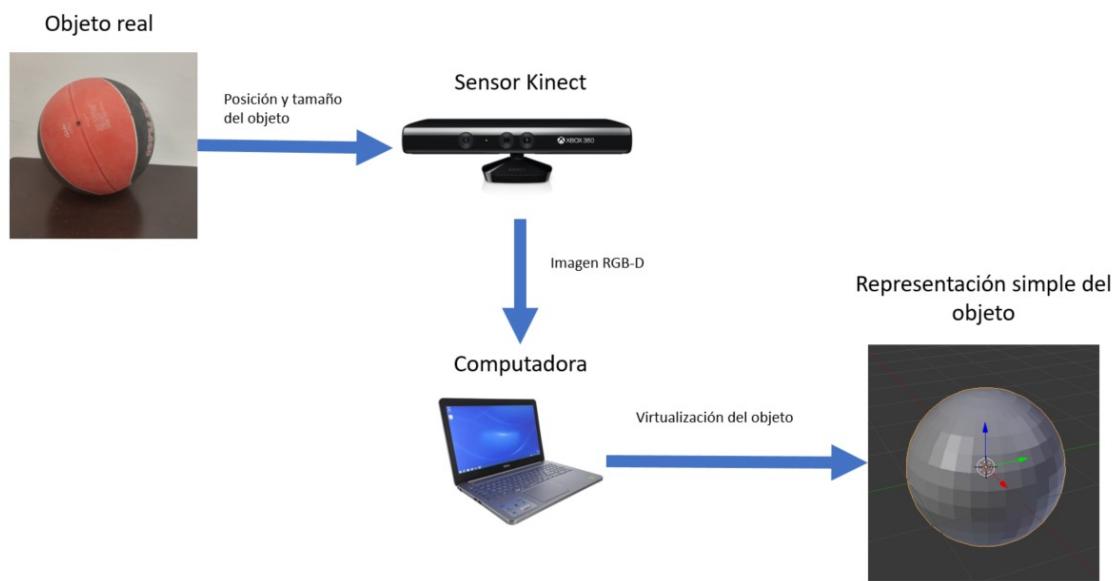


Figura 1.12: Diagrama del sistema propuesto

2

DESARROLLO

2.1 METODOLOGÍA

En esta sección se describe como fue el desarrollo del sistema, las herramientas usadas, así como detalles de la implementación.

Se desarrolló el sistema usando la metodología iterativa incremental. Esta metodología permite el desarrollo de un sistema incrementando sus capacidades cada iteración. En cada iteración se analiza, desarrolla y prueba un objetivo diferente, si al final de las pruebas el objetivo se cumplió se guarda el proyecto desarrollado y se empieza a trabajar con el siguiente objetivo, si el proyecto no pasa las pruebas se desecha y se vuelve a analizar.

En la figura 2.1 se muestra el diagrama a bloques del sistema a desarrollar, el primer bloque es la captación de los datos de un objeto, el segundo el clasificador que hace uso del AGC para obtener la representación del objeto.

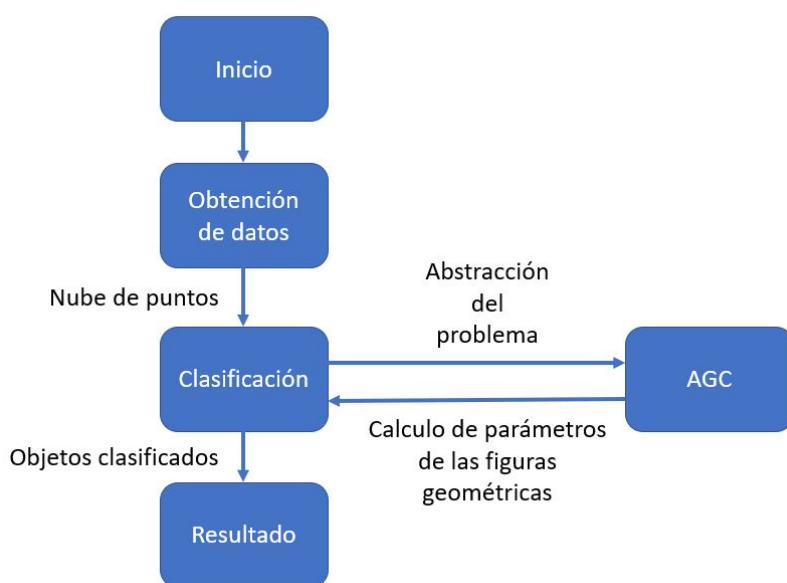


Figura 2.1: Diagrama a bloques del sistema

Como primer objetivo se planteó la obtención de datos del Kinect, luego la clasificación de los objetos y por último la integración del AGC.

2.2 PRE-PROCESAMIENTO

A continuación se describen las herramientas y métodos usados en el pre-procesamiento para obtener los datos del sensor Kinect, crear la nube de puntos, y separar la nube de puntos en nubes más pequeñas que contienen los objetos a representar.

2.2.1 Biblioteca de Nubes de Puntos(PCL)

Para realizar el manejo de **nubes de puntos** se uso la biblioteca de nubes de puntos (PCL por sus iniciales en inglés)[4], la cual cuenta con un conjunto de herramientas comúnmente usadas para quitar ruido, agrupar puntos, segmentación, etc.



Figura 2.2: Logo de la Biblioteca de Nube de Puntos

PCL es un proyecto de código abierto. El logotipo del proyecto se puede ver en la figura 2.2, desarrollado y apoyado por grandes empresas, este proyecto se distribuye bajo la licencia BSD modificada (de 3 cláusulas). Se ha probado en sistemas Linux, MacOS y Windows, además de que el código esta creado de forma modular lo que permite elegir que características usar y cuales descartar.

2.2.2 Obtención de datos

La obtención de una nube de puntos puede ser dada por una variedad de **sensores**, estos se clasifican según la tecnología usada para obtener la imagen de profundidad, algunos ejemplos son los sensores: ultrasónicos, infrarrojos, estereoscópicos. En este caso particular se utiliza el sensor de Microsoft Kinect 2.0, ya que es uno de los dispositivos que generan imágenes de profundidad con mayor soporte y de los más conocidos, ademas con un precio de \$2,699.00 MXN [23], es un dispositivo bastante accesible.

Para generar la nube de puntos a usar es necesario un intermediario entre PCL y el Kinect ya que los datos del Kinect no se encuentran listos para ser usados directamente con la bibliotecas de PCL, ya que la información obtenida del Kinect son tres imágenes, de color, infrarroja y de profundidad, como se muestra en la figura 2.3 [5], estas imágenes se combinan para construir una nube de puntos, primero se juntan las imágenes de color y la de profundidad y luego se genera la nube de puntos la cual representa cada pixel de la imagen de profundidad en un arreglo de puntos junto con el color asignado a cada uno.



Figura 2.3: Datos obtenidos del dispositivo Kinect, (a) Imagen de la cámara a color, (b)Imagen de la cámara infrarroja, (c)Imagen de profundidad.

El [controlador libfreenect](#) [24] se comunica con el Kinect y obtiene las imágenes a color RGB, infrarrojas (IR) y profundidad. Este proyecto fue el resultado del trabajo de H. Martín para el concurso que lanza Adafruit [5]

Mientras [libfreenect2pclgrabber](#) [25] es un proyecto desarrollado por Dr. Giacomo Dabisias ingeniero de software en lyft [25], que cuenta con una biblioteca que actúa como intermediario creando nubes de puntos a partir de las imágenes infrarrojas, de profundidad y color obtenidas del controlador libfreenect. Se uso este proyecto de software libre ya que cuenta con soporte para sistemas operativos Linux, MacOs y Windows, permitiendo la reproducción e integración del sistema desarrollado en diferentes sistemas operativos.

Luego de que el controlador obtiene los datos del sensor y son convertidos en una nube de puntos es posible empezar a trabajar con ellos usando las herramientas de PCL.

2.2.3 Filtrado de densidad de puntos

Dado que la nube de puntos obtenida cuenta con muchos datos, el procesamiento se vuelve lento. Para disminuir la carga de trabajo de la computadora lo primero que se realiza es un filtrado de densidad de puntos, dicho de otra manera se busca disminuir la cantidad de puntos en la nube, parecido a cuando se reduce el tamaño

de una imagen.

Este **filtro** calcula una nueva nube de puntos la cual contiene menos información de la original, un ejemplo de este filtro se puede ver en la figura 2.4. cada uno de los nuevos puntos es una representación de un conjunto de puntos que se encuentran dentro de alguna vecindad.

Los pasos para este filtro son:

- El filtro divide la nube de puntos en un arreglo de cubos.
- Luego se calcula el centroide de cada división usando todos los puntos que se encuentran dentro de esta.
- Al final se crea una nueva nube de puntos usando los centroides obtenidos.



Figura 2.4: Filtro de densidad de puntos, (a) Nube de puntos obtenida del sensor Kinect, (b) Nube de puntos luego del filtrado de densidad.

2.2.4 Operación Resta

Para detectar los objetos se propuso realizar una comparación entre dos nubes de puntos como se muestra en la figura 2.5, en la figura 2.5a, se muestra la nube que representa los puntos de una escena con la cual no se va a interactuar ni cambiar de lugar ningún objeto, en la figura 2.5b, se muestra la nueva nube de puntos captada con los objetos a representar. Al comparar las nubes de puntos la operación resta muestra los cambios entre la escena estática y la actual, resaltando solo los objetos de interés, como se muestra en la figura 2.5c. Esta operación es similar a la operación

resta entre imágenes, con la diferencia de que se realiza en tres dimensiones sobre nubes de puntos.

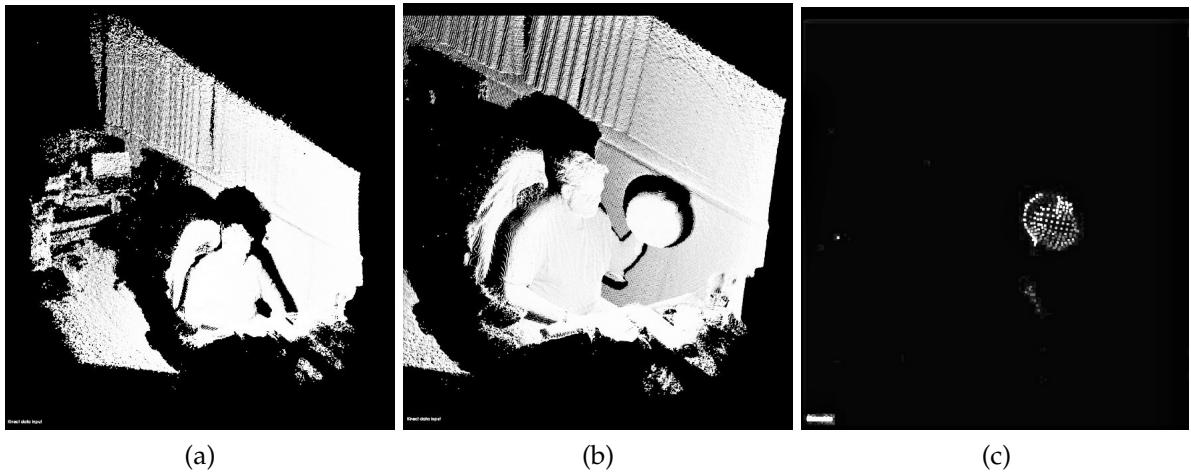


Figura 2.5: Operación resta, (a) Escena estática, (b) Escena nueva con objetos., (c) Nube de puntos luego de la operación resta.

2.2.5 Agrupamiento

Teniendo la nube de puntos con únicamente los puntos de los objetos que nos interesan, ahora lo que se hace es separar en nubes cada objeto. Para esta tarea se realiza una operación llamada Clustering, que traducida se puede interpretar como agrupar, en este caso se usa para agrupar los puntos cercanos.

El proceso de agrupamiento se muestra en el [algoritmo 2.1](#).

Algoritmo 2.1: Agrupamiento

```

1 begin
2   while N contenga puntos do
3     NN=Nueva nube de puntos
4     NN.add(N[o])
5     N.remove(N[o])
6     foreach P in NN do
7       foreach p in N do
8         if Distancia de p a P ; intervalo then
9           NN.add(p) N.remove(p)
10    LN.add(NN)

```

Donde:

- P y p son puntos
- N es la nube de puntos
- NN es una nueva nube de puntos que contiene un objeto

Al final se obtiene un arreglo de nubes y cada nube contiene los datos de un solo objeto, y así trabajar con cada objeto por separado.

2.3 CLASIFICACIÓN DE OBJETOS

2.3.1 Transformada de Hough

La transformada de Hough es usada para encontrar líneas y circunferencias en una imagen, como se ve en la figura 2.6. En el trabajo Extending Generalized Hough Transform to Detect 3D Objects in Laser Range Data se plantea una modificación de la transformada de Hough para usarla en espacios 3D con nubes de puntos y en lugar de buscar líneas encontrar cilindros.

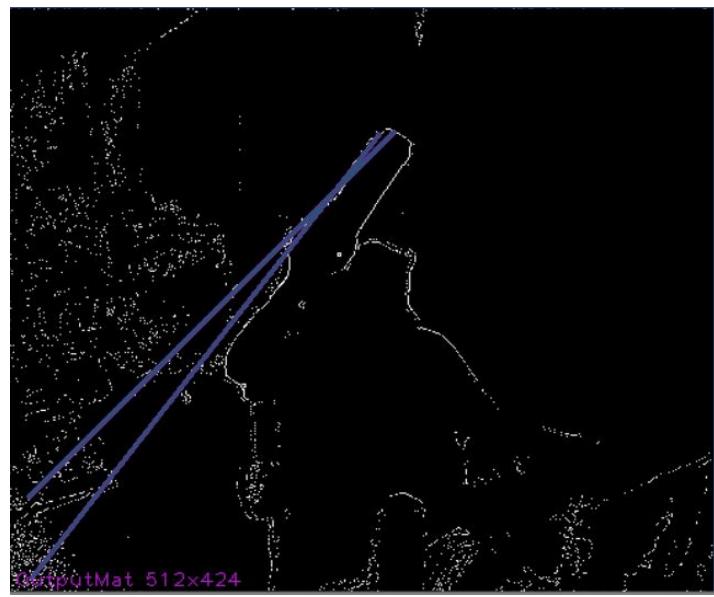


Figura 2.6: Prueba de la transformada de Hough

Este método se descartó ya que al programar de la transformada simple de Hough en Unity3D tardo en calcular las líneas de una escena dada por el Kinect entre diez a quince segundos, y como se busca desarrollar un sistema que permita la interacción continua con el usuario el tiempo de respuesta debe ser lo más rápido posible para

que la información se obtenga al momento sin tener que esperar.

2.3.2 Red Neuronal

El método anterior se descartó y analizó el problema desde otra perspectiva. Un método que ha dado buenos resultados en el área de visión por computadora son las redes neuronales.

Una red neuronal es una abstracción de como se cree funciona el cerebro, teniendo como base una neurona la cual tiene estímulos de entrada y dependiendo de una función de activación envía una señal de salida. La función de activación depende de las señales de entrada y valores que se ajustan de manera automática en el entrenamiento.

Las redes neuronales son capaces de realizar tareas de reconocimiento de manera rápida y bastante acertada, la desventaja se encuentra en el proceso de aprendizaje también llamado entrenamiento, para que la red neuronal pueda reconocer un objeto, la red debe estar entrenada para reconocer dicho objeto. Eso crea el problema de generar una base de datos de cada objeto a usar, luego realizar el entrenamiento.

Dado a que no se cuenta con una base de datos de los objetos que se quieren representar y lo complicado que seria para el usuario agregar nuevos objetos, este método también fue descartado.

2.3.3 RANSAC

Como se vio en la introducción el algoritmo RANSAC se usó para estimar los parámetros de los modelos geométricos, se definieron los modelos, y los parámetros del algoritmo. Este método es preciso en la estimación de los valores de los objetos, permite estimar el tamaño, orientación y posición del objeto modelado.

Para atacar el problema primero se implementó RANSAC para detectar esferas como se muestra en el [algoritmo 2.2](#).

Donde:

- nuevoObjeto - Nube de puntos obtenida del pre-procesamiento.
- n - Número mínimo de puntos para crear la esfera (4 puntos)
- k - Número máximo de iteraciones permitidas.
- t - Valor del intervalo de error permitido para determinar si un dato pertenece al modelo.

Algoritmo 2.2: RANSAC

```

1 begin
2   k = numeroRepeticiones iteraciones = 0
3   n = 4 mejorModelo = null
4   mayorPuntosEnLaEsfera = null
5   while iteraciones < k do
6     puntosAleatorios[n] = DatosAleatoriosDe(nuevoObjeto)
7     nuevaEsfera = GeneraNuevaEsfera(puntosAleatorios)
8     puntosEnLaEsfera = []
9     foreach punto in nuevaEsfera do
10       if punto.PerteneceA(nuevaEsfera, error < t) then
11         puntosEnLaEsfera.Add(punto)
12       if mayorPuntosEnLaEsfera.Length() < puntosEnLaEsfera.Length() then
13         mejorModelo = nuevaEsfera
14         mejorPuntosEnLaEsfera = puntosEnLaEsfera
15       iteraciones++
16   return mejorModelo

```

En el pseudocódigo las variables *mejorModelo* y *mayorPuntosEnLaEsfera* guardan la descripción del mejor modelo obtenido por el método RANSAC y la lista de los puntos que pertenecen a esa descripción del modelo, se inician con el valor *null*, para evitar confusiones con los otros valores, ya que *null* $\neq 0$, la variable *iteraciones* encargada de contar el numero de ciclos realizados se inicia en 0, y el valor de *k* esta dado por el numero de repeticiones que necesarias para encontrar una buena estimación del modelo, el cual se calcula mas adelante.

En este caso se uso el modelo de una *esfera* en AE la cual usa cuatro puntos para describirla, el modelo cuenta con cuatro variables a definir (x, y, z, r) , que corresponden al centro de la esfera y su radio, aquí es donde se obtiene el valor de la variable *n*, que esta dado por la cantidad de puntos en *puntosAleatorios* que son el conjunto mínimo puntos necesarios para encontrar los valores para (x, y, z, r) , se calcula el modelo resultante *nuevaEsfera* usando la ecuación (ec. 9).

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2 \quad (9)$$

Para el *plano* en AE se usa la forma normal de Hesse el cual consiste en la ecuación (ec. 10), donde *n* es el vector normal al plano y *d* es la distancia que hay del origen al plano, el signo de *d* también determina en que dirección de *n* se encuentra el origen, si *d* > 0 entonces el origen se encuentra en la dirección de *n*, si *d* < 0 el origen se encuentra en dirección contraria a *n*.

$$n \cdot x = -d \quad (10)$$

Por último para el [cilindro](#) se usa la construcción mostrada en la ecuación (ec. 11), usando un punto base $x = (x, y, z)$ por el cual pasa un eje en dirección del vector $n = (x, y, z)$ y un radio r .

$$C = \{x, n, r\} \quad (11)$$

Los puntos *puntosAleatorios* son elegidos de forma aleatoria y el modelo resultante *nuevaEsfera* es comparada con cada punto en la nube de puntos para ver cuales puntos cumplen con la descripción del modelo, en otras palabras se revisa si el punto pertenece a la superficie del objeto, si es así, el punto se agrega a la lista *puntosEnLaEsfera*.

La lista resultante se compara con la lista generada por el mejor modelo encontrado, para el primer ciclo, como aún no se tiene un modelo anterior para comparar, el modelo y la lista tienen un valor *null*, por lo cual el primer modelo calculado se guarda como el mejor modelo, que posteriormente será remplazado por alguno mejor.

Para calcular el número de repeticiones k necesarias para elegir un buen conjunto de puntos n , siendo w la probabilidad de que un punto seleccionado pertenece a una buena aproximación del modelo y su tolerancia de error, obtenemos la ecuación (ec. 12).

$$E(k) = b + 2 * (1 - b) * b + 3 * (1 - b)^2 * b \dots + i * (1 - b)^{i-1} * b + \dots \quad (12)$$

$$E(k) = b * [1 + 2 * a + 3 * a^2 \dots + i * a^{i-1} + \dots] \quad (13)$$

Donde $E(k)$ es el valor esperado de k , $b = w^n$, y $a = (1 - b)$. aplicando la identidad para series de sumas geométricas (ec. 14).

$$a / (1 - a) = a + a^2 + a^3 \dots + a^i + \dots \quad (14)$$

realizando la derivada con respecto a a , obtenemos la ecuación (ec. 15).

$$a / (1 - a)^2 = 1 + 2 * a + 3 * a^2 \dots + i * a^{i-1} + \dots \quad (15)$$

obtenemos (ec. 16)

$$E(k) = 1/b = w^{-n} \quad (16)$$

En la tabla 2.1 se muestran los valores para $E(k)$ variando los valores de n y w , siendo $n = 4$ para la esfera y $w = 0.2$ serían necesarias al menos 625 repeticiones.

De igual manera se busca exceder el valor de $E(k)$ por una o dos derivaciones estándar. La derivación estándar $SD(k)$ esta dada por la ecuación (ec. 17)

$$SD(k) = \sqrt{E(k^2) - E(k)^2} \quad (17)$$

Tabla 2.1: Valores calculados de E(k)

w	n=1	2	3	4	5	6
0.9	1.1	1.2	1.4	1.5	1.7	1.9
0.8	1.3	1.6	2.0	2.4	3.0	3.8
0.7	1.4	2.0	2.9	4.2	5.9	8.5
0.6	1.7	2.8	4.6	7.7	13	21
0.5	2.0	4.0	8.0	16	32	64
0.4	2.5	6.3	16	39	98	244
0.3	3.3	11	37	123	412	-
0.2	5.0	25	125	625	-	-

Para calcular $E(k^2)$ se usa la ecuación (ec. 18) que se obtiene de un proceso similar al usado anteriormente usando la derivación de una identidad.

$$E(k^2) = \frac{2 - b}{b^2} \quad (18)$$

Y $SD(k)$ queda como en la ecuación (ec. 19), que para la esfera se obtiene un valor de $SD(k) = 625.499 \simeq 626$, dando un valor total de repeticiones para el cálculo del modelo de la esfera de $k = E(k) + SD(k) = 625 + 626 = 1,250$.

$$SD(k) = [\sqrt{1 - w^n}] * (1/w^n) \quad (19)$$

Otra forma de decidir la cantidad de repeticiones necesarias para encontrar el modelo se basa en la probabilidad. De esta forma se asegura que al menos una de las selecciones aleatorias entra en el rango de tolerancia de la predicción. Este método es el usado para el sistema ya que el número de iteraciones es menor o igual al método anterior.

La probabilidad de encontrar el modelo deseado eligiendo q puntos aleatorios se muestra en ecuación (ec. 20).

$$P(n) = \frac{\binom{n}{q}}{\binom{N}{q}} \approx \left(\frac{n}{N}\right)^q \quad (20)$$

Donde N es la cantidad de puntos en la nube de puntos, n los puntos que pertenecen al modelo.

La probabilidad de encontrar el modelo luego de k iteraciones está dada por $P_{(n,k)}$ que se muestra en la ecuación (ec. 21).

$$P(n, k) = 1 - (1 - P(n))^k \quad (21)$$

Despejando k obtenemos la ecuación (ec. 22).

$$K \geq \frac{\ln(1 - P_{(n,k)})}{\ln(1 - P(n))} \quad (22)$$

La ecuación (ec. 22) es calculada cada que se encuentra un mejor modelo. Entre mejor sea la representación del modelo deseado el valor k será menor. El realizar el cálculo de k permite terminar al algoritmo RANSAC antes de la cantidad de repeticiones estimadas con el método anterior y agilizar el algoritmo, la modificación se observa en el [algoritmo 2.3](#) en las líneas [15-17].

Algoritmo 2.3: RANSAC modificado

```

1 begin
2     itreciones=0
3     mejorModelo=null
4     mayorPuntosEnLaEsfera = null
5     while itreciones< k do
6         puntosAleatorios[n] = DatosAleatoriosDe(nuevoObjeto)
7         nuevaEsfera = GeneraNuevaEsfera(puntosAleatorios)
8         puntosEnLaEsfera = []
9         foreach punto in nuevoObjeto do
10            if punto.PerteneceA(nuevaEsfera, error<t) then
11                puntosEnLaEsfera.Add(punto)
12            if mayorPuntosEnLaEsfera.Length()< puntosEnLaEsfera.Length() then
13                mejorModelo = nuevaEsfera
14                mejorPuntosEnLaEsfera = puntosEnLaEsfera
15                w = mejorPuntosEnLaEsfera/nuevoObjeto.Length()
16                z = 0.99
17                k = log( 1.0 - z )/ log( 1.0 - pow( w, n))
18            iteraciones++
19     return mejorModelo

```

El valor de z viene preestablecido en las bibliotecas de PCL con el valor 0.99, este valor puede ser modificado, pero al obtener un resultado aceptable se decidió no modificar los valores.

2.4 DESCRIPCIÓN GENERAL DEL SISTEMA

Para comprender el desarrollo de este [sistema](#) en la figura 2.7 se muestra el diagrama de flujo que describe el comportamiento del sistema. Se observa que son importar que datos de obtengan el sistema siempre realiza ocho procesos, los primeros cuatro procesos corresponden al pre-procesamiento en el cual se obtienen, filtran y separan los datos obtenidos del Kinect, luego se observa el cálculo de los modelos (esfera, plano y cilindro), ya sea usando AE o AGC el método se realiza de la misma manera para luego compararlos dependiendo de cual describe mejor a al conjunto de datos obtenidos, para el uso de AGC los puntos fueron transformados del espacio euclidiano \mathbb{R}^3 al espacio $\mathbb{R}^{n+1,1}$, así como las figuras obtenidas usando AGC fueron interpretadas para su representación en el espacio euclidiano.

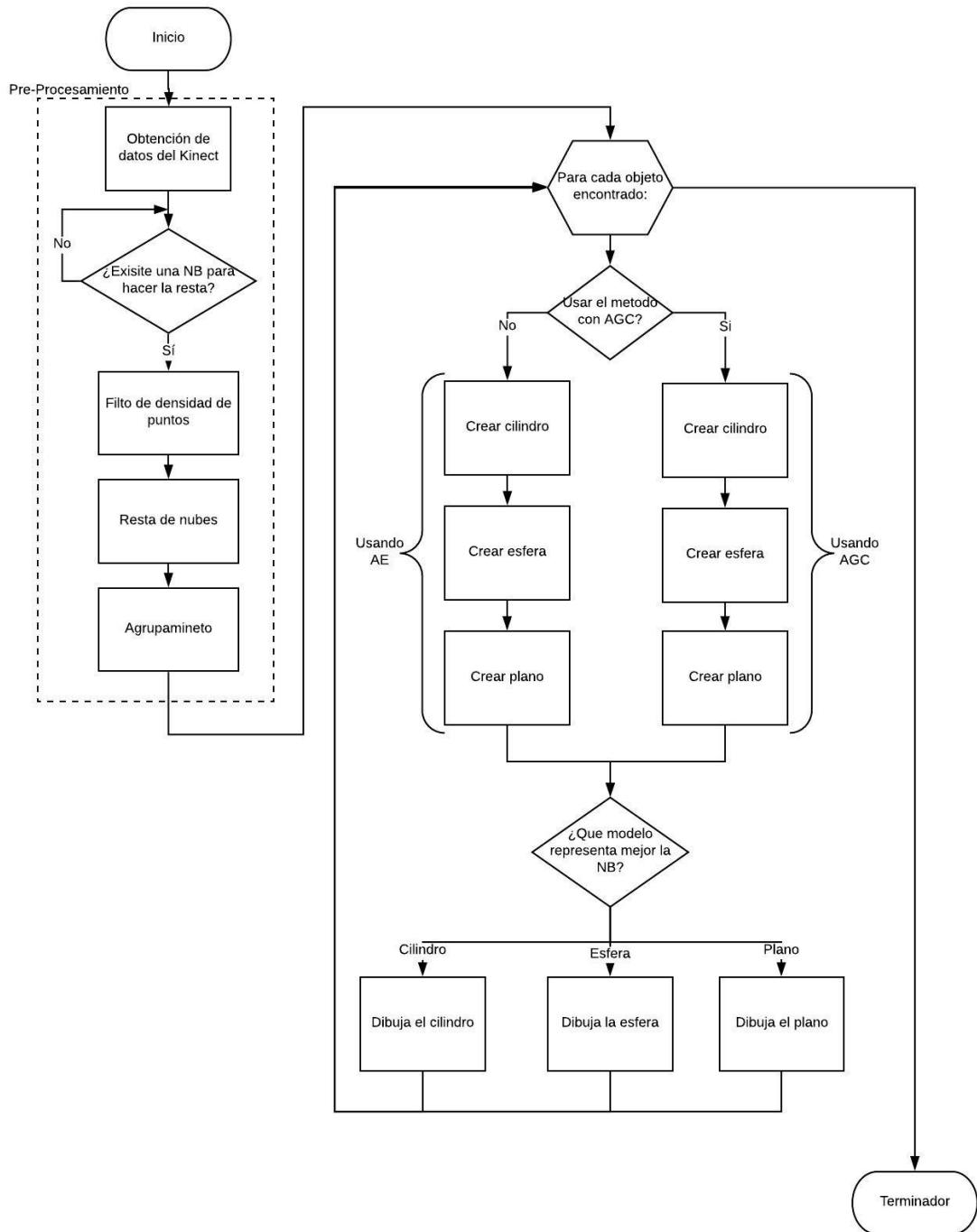


Figura 2.7: Diagrama de flujo del sistema

2.5 ÁLGEBRA GEOMÉTRICA CONFORME

En la sección pasada se describió la estimación de los modelos usando RANSAC. En esta sección se presentan modelos matemáticos para el plano, esfera y cilindro en

AGC y las funciones usadas para validar los puntos del modelo.

Las entidades en AGC cuentan con dos representaciones (IPNS y OPNS), ya que es fácil pasar de una forma a otra se puede trabajar con cualquiera de ellas dependiendo de las necesidades.

Una **esfera** en AGC en IPNS se representa como se observa en la ecuación (ec. 23), donde P es el centro en AGC y r el radio de la esfera, y en OPNS como la ecuación (ec. 24), donde P_i son puntos de AGC que se encuentran en la superficie de la esfera.

$$S = P - \frac{1}{2}r^2 e_{\infty} \quad (23)$$

$$S^* = P_1 \wedge P_2 \wedge P_3 \wedge P_4 \quad (24)$$

Ya que para el algoritmo de RANSAC hay que crear un modelo, esto se realiza de forma similar que para una esfera AE usando cuatro puntos. lo único que falta para crear el modelo es escribir un punto de AE en AGC para eso se usa la ecuación (ec. 25), donde \mathbf{x} es el vector (x, y, z) .

$$P = \mathbf{x} + \frac{1}{2}\mathbf{x}^2 e_{\infty} + e_0 \quad (25)$$

La forma de validar si un punto se encuentra sobre la superficie de la esfera es diferente al usado en AE ya que no hay forma directa para calcular la distancia mas corta, sin embargo, es posible calcular la distancia del punto a la tangente de la esfera como se muestra en la figura 2.8 usando la ecuación (ec. 26), donde s es el vector (S_1, S_2, S_3) y p es el vector (P_1, P_2, P_3) [11].

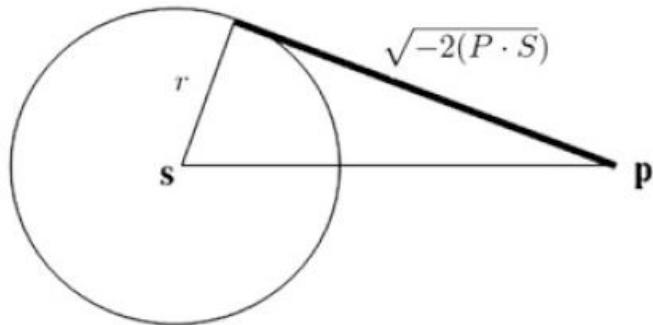


Figura 2.8: El producto interno de un punto y una esfera describe el cuadrado de la distancia entre el punto y la tangente de la esfera.

$$P \cdot S = \frac{1}{2}r^2 - \frac{1}{2}(s - p)^2 \quad (26)$$

De tal manera que la distancia $\sqrt{-2(P \cdot S)}$ entre el punto y el punto tangente a la esfera esta dado por la ecuación (ec. 27)

$$-2(P \cdot S) = (s - p)^2 - r^2 \quad (27)$$

Para encontrar el modelo del [plano](#), se realiza de manera similar a la esfera, primero hay que convertir los puntos usando la ecuación (ec. 25), y usar la representación de un plano en forma OPNS que se muestra en la ecuación (ec. 28), donde P_i son puntos en la superficie del plano, y la forma IPNS esta dada por la ecuación (ec. 29), donde \mathbf{n} es el vector normal del plano y d la distancia del origen al plano sobre el vector normal n .

$$\pi^* = P_1 \wedge P_2 \wedge P_3 \wedge e_\infty \quad (28)$$

$$\pi = \mathbf{n} + d e_\infty \quad (29)$$

Determinar si los puntos se encuentran dentro de la superficie del plano, se usa la ecuación (ec. 30) que determina la distancia entre el punto y el plano, donde π^* es el plano en la forma OPNS, P es el punto en AGC, p es el vector (x, y, z) de P , n es el vector normal al plano y d la distancia del origen al plano.

$$P \cdot \pi^* = p \cdot n - d \quad (30)$$

Para la creación del [cilindro](#) es un poco más compleja, ya que no se cuenta con una identidad para crear el cilindro es necesario construirlo usando el modelo creado a partir de dos esferas descrito por Aksel Sveier [19].

El cilindro se crea con un eje y un radio, para encontrar el eje se buscan dos esferas usando el método RANSAC y trazando una línea entre sus centros se obtiene el eje del cilindro, como se muestra en la figura 1.11b, para obtener el radio del cilindro se promedian los radios de las esferas.

La construcción de una línea que pasa por el centro de dos esferas se calcula usando la ecuación (ec. 31), donde S_i^* es una esfera en la forma OPNS.

$$L = S_1^* \wedge S_2^* \wedge e_\infty \quad (31)$$

La forma de saber si un punto pertenece al cilindro es creando una esfera cuyo centro pasa por el eje del cilindro y el ecuador de la esfera pasa por el punto a evaluar, si el radio de la esfera coincide con el radio del cilindro el punto pertenece al cilindro.

La construcción de la esfera necesaria para conocer si un punto pertenece a un cilindro esta dada por la ecuación (ec. 32), donde P es el punto en AGC a evaluar, L^* es la línea que pasa por el eje del cilindro.

$$S_1 = \frac{P \wedge L^*}{L^*} \quad (32)$$

Para obtener el radio r de una esfera en AGC se usa el producto interno de la esfera por si misma, como se muestra en la ecuación (ec. 33).

$$r = \sqrt{S_1 \cdot S_1} \quad (33)$$

2.5.1 Herramientas para el uso del AGC

Para la experimentación con el AGC existen una variedad de herramientas listadas a continuación:

- Cinderella [26]
- GAViewer [27]
- CluCalc/CluViz [28]
- Gaalop [29]
- Vedor [30]
- Vedor.js [31]

Se uso la herramienta GAViewer como se muestra en la figura 2.9 para experimentar y validar operaciones en AGC. GAViewer es un programa multipropósito para el computo de operaciones en AG creado por Daniel Fontijne, Leo Dorst, Tim Bouma de la universidad de Amsterdam y Stephen Mann de la universidad de Waterloo, cuenta con una interfaz gráfica y entrada por comandos en los cuales se pueden trabajar con diferentes AG así como AGC, permite la conexión de otros programas para realizar las operaciones de AG y visualizar los resultados. El uso de este programa para un desarrollo robusto es limitado ya que el lenguaje de programación usado es lento y con limitaciones[27].

Para el desarrollo del sistema se uso la biblioteca Vedor [30], el cual cuenta con soporte para AG, AE, AGC, entre otro tipos de álgebras, permite modificar y acoplar las funciones al sistema y evitar tener que establecer conexiones con otros programas como con el caso de GAViewer. La biblioteca cuenta con funciones matemáticas así como funciones para la creación de gráficos usando OpenGL y OpenGles

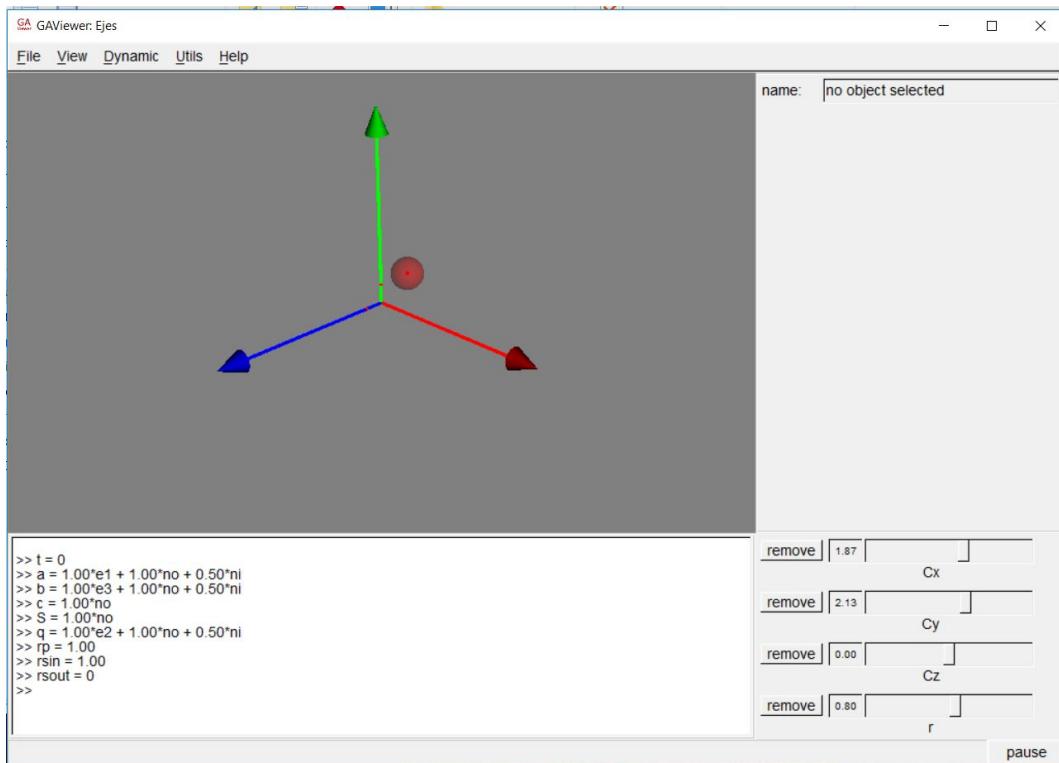


Figura 2.9: Programa GAViewer

3

PRUEBAS Y CASO DE ESTUDIO

Para comprobar el funcionamiento del sistema se realizaron dos pruebas de caja negra y un caso de estudio para evaluar su desempeño, las pruebas de caja negra son efectuadas sobre módulos del sistema los cuales tienen datos de entrada, la funcionalidad del módulo y la salida. Mientras que un caso de estudio ayuda a examinar y comparar los métodos usados en el sistema. cada prueba y el caso de estudio busca la solución de los objetivos de este trabajo.

3.1 PRUEBA DE CAPTURA DE DATOS

Ya que el primer objetivo específico es la realización del módulo del sistema encargado de la obtención de datos, en esta sección de pruebas se enfoca al tipo de entradas que permite el sistema, para esto se realizó una prueba de caja negra con la cual se establecen criterios de entrada para el tipo de objetos que pueden ser.

En esta prueba se buscan los objetos que no son admitidos por el sistema, para esto se uso sólo una población representativa de diferentes casos en los que los objetos no serían admitidos.

Los objetos no admitidos se dividen por tamaño y material.

3.1.1 *Por tamaño*

El sistema sólo reconoce objetos que generen al menos veinte puntos, Dado que el Kinect da una densidad de puntos cercana a un punto por centímetro a un metro de distancia como se muestra en la figura 3.1, y el sistema realiza un filtrado de puntos obteniendo un punto cada dos y medio centímetros. Los objetos deben de contar con un área mínima visible al Kinect cincuenta centímetros cuadrados.

Por el otro lado, el objeto no necesita ser visto en su totalidad por el Kinect, pero la clasificación solo se realizará con los datos que puedan ser captados por el Kinect.

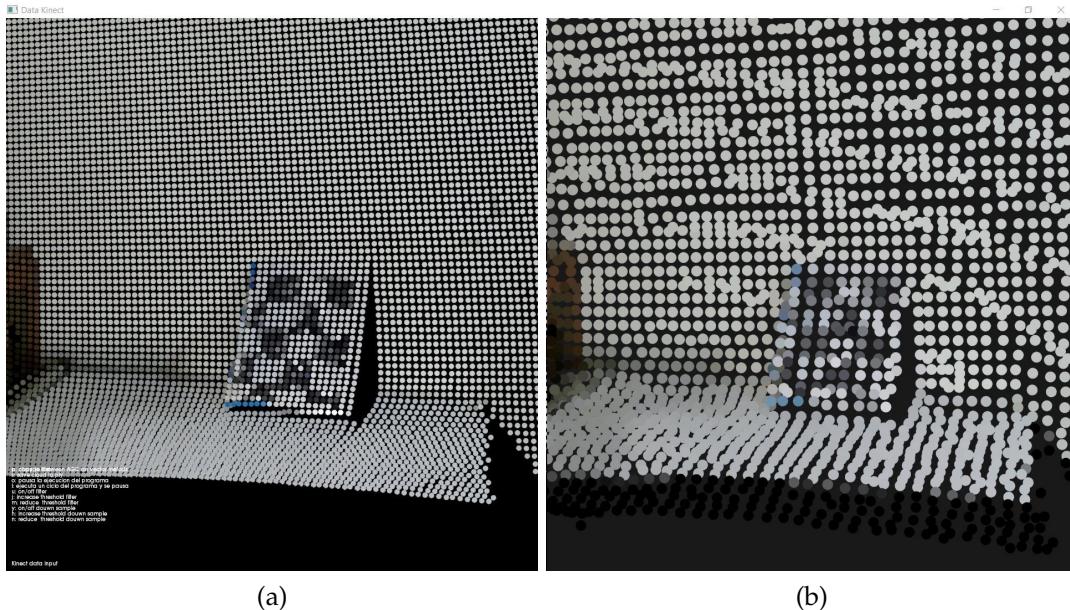


Figura 3.1: Prueba para el filtro de densidad de puntos, (a) Nube de puntos obtenida del Kinect sobre una cuadrícula de 5cm, (b) Nube de puntos resultante luego del filtro de densidad.

3.1.2 *Por material*

El Kinect al realizar la detección de profundidad usando ToF depende de la luz infrarroja, si un objeto cuenta con un material que absorbe este tipo de luz el Kinect no será capaz de determinar la nube de puntos. de forma similar ocurre con materiales reflectantes y translúcidos.

3.2 PRUEBA DE SEGMENTACIÓN

Para el módulo encargado de la segmentación se realizó de una prueba de caja negra para encontrar el espacio de trabajo del sistema. Esta prueba engloba tanto la resta de nubes como el agrupamiento de estos.

3.2.1 *Escenario*

Ya que para la segmentación se realiza una resta, es necesario capturar el escenario en el cual trabaja el sistema. Este escenario debe cumplir con una serie de lineamientos:

- No debe estar expuesto a la luz del sol o algún otro emisor de luz infrarroja.
- El escenario no debe modificarse mientras el sistema este trabajando.
- El escenario no debe contener objetos reflectantes o translúcidos.

- El escenario debe estar separado del Kinect 0.5m y no estar más alejado de 4.5m.

3.2.2 *Objetos*

- El área visible al Kinect debe sobresalir al menos 2.5cm sobre el escenario.
- Los objetos deben estar despegados más de 3cm.

3.3 DESCRIPCIÓN DEL CASO DE ESTUDIO

Se planteo el caso de estudio para comparar los dos sistemas propuestos, se busca conocer que método brinda la mejor precisión del sistema para representar correctamente los objetos, y la exactitud al representar cada uno de ellos. Ya que el sistema representa tres tipos de modelos diferentes, se realizo la comparación de los datos para cada uno de los modelos, siendo cada modelo una variable distinta en el caso de estudio, se tienen tres variables, de esta manera se evalúa el sistema dependiendo del modelo esperado, y se obtiene la comparación de los dos métodos para cada modelo propuesto.

Antes de continuar hay que aclarar como es que se esta haciendo uso de los conceptos de precisión y exactitud.

Para este caso se entiende como precisión a la capacidad del sistema para realizar la clasificación del objeto correctamente, es decir, si el sistema se le presenta el mismo objeto la precisión esta dada por la probabilidad de obtener el mismo tipo de figura en cada ocasión. El sistema es evaluado usando objetos similares a los modelos planteados, cada objeto es evaluado de forma independiente, se le presenta el objeto y se obtiene la decisión tomada por el sistema, este evento es repetido cien veces para obtener la precisión P , dada por la división entre la cantidad de veces que el sistema acertó entre el total de intentos del sistema como se muestra en la ecuación (ec. 34).

$$P = \frac{\#Aciertos}{\#Intentos} \quad (34)$$

Mientras que exactitud E se entiende como la capacidad del sistema para encontrar el modelo que mejor representa a la nube de puntos del objeto. Para evaluar la exactitud la prueba se realiza solo cuando el sistema ha seleccionado de manera adecuada el tipo de modelo a usar, si el objeto usado en la evaluación es un balón se obtiene la exactitud solo cuando el sistema decide representarlo como una esfera. La exactitud se obtiene con la ecuación (ec. 35) donde $P.Modelo$ es la cantidad de puntos que coinciden con el modelo y $P.Objeto$ es el total de puntos del objeto.

$$E = \frac{P.Modelo}{P.Objeto} \quad (35)$$

La obtención de los datos para al evaluación del caso de estudio se realiza de forma similar para cada modelo, por eso solo se explica como se realiza para el caso de la [esfera](#), y esto se repite para los modelos del [plano](#) y el [cilindro](#), variando únicamente los objetos usados, para la esfera se uso un balón, para el plano una carpeta de cartón, y para el cilindro una botella de agua. Se eligieron estos objetos ya que presentan una geometría similar a la propuesta, esto representa un caso especial para el sistema ya que con estos objetos se determina que tan capaz es el sistema de excluir a los otros modelos, y se esperaría que el sistema obtenga una capacidad de clasificación por arriba de un 60% al clasificar los objetos, para saber que no son resultados obtenidos de forma aleatoria .

Se coloco el sensor Kinect apuntando a una mesa como se muestra en la figura [3.2](#). Para el caso de estudio este es el escenario con el cual no se interactúa y que es filtrado usando la resta de nubes de puntos.



Figura 3.2: Estudio de pruebas

Se inicia el sistema, y se obtiene la nube de puntos del escenario, luego se coloca el balón sobre la mesa, como se observa en la figura [3.3](#), el sistema realiza el pre-procesamiento y la estimación de los modelos usando RANSAC con los métodos de la biblioteca PCL, se obtienen cien estimaciones del sistema para el balón, y se repite para obtener la estimación usando RANSAC con AGC.

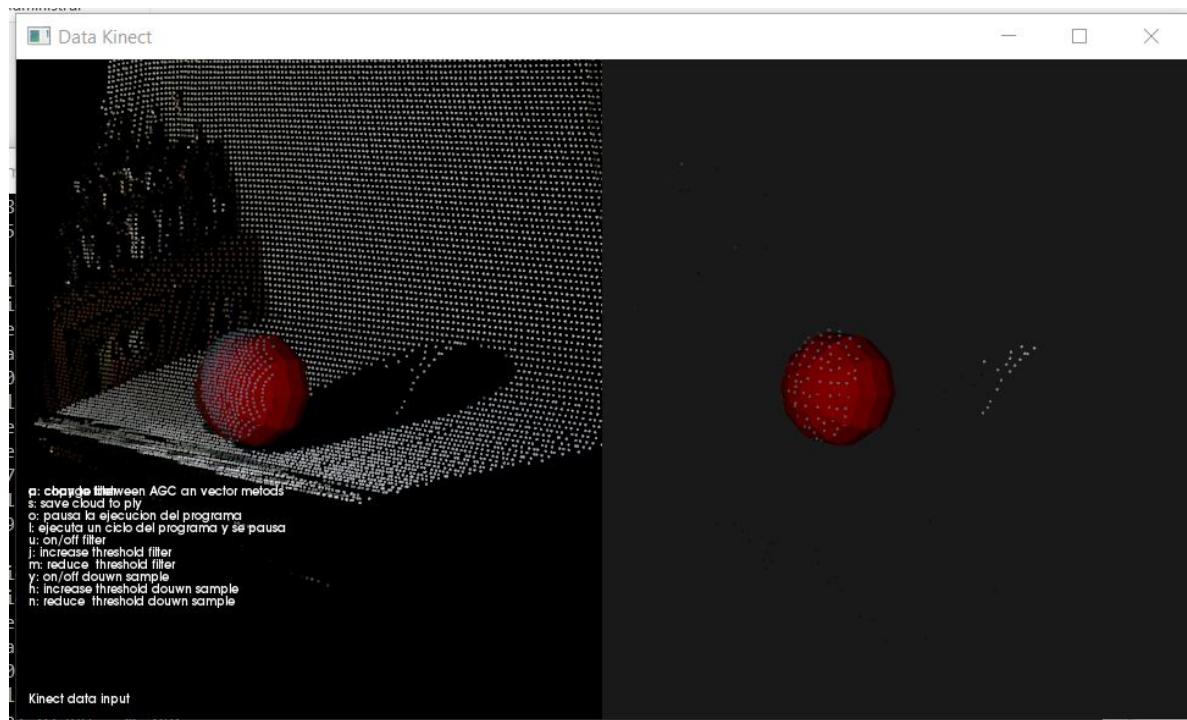


Figura 3.3: Prueba para la esfera

Para establecer si los datos obtenidos son diferentes y no eventos aislados se realiza la prueba de suma de rangos de Wilcoxon para datos independientes, para esta prueba se seleccionan los primeros treinta casos en los que el sistema seleccionó de manera acertada el tipo de modelo a usar.

La prueba de suma de rangos de Wilcoxon se usa para comparar la distribución de grupos de datos, es una prueba no paramétrica, es decir que no se conoce el tipo de distribución de los datos o la distribución de los datos no es normal, siendo la prueba z y la t de Student para datos independientes pruebas equivalentes para datos paramétricos, y es para datos independientes ya que los resultados para un método son independientes de los resultados del otro.[32].

Aplicando la prueba de sumas de rangos de Wilcoxon en el caso de estudio para comparar la exactitud para el modelo de la esfera, primero se establece la hipótesis, ya que se espera una diferencia entre los métodos usados la hipótesis esperada es H_1 : Los métodos varían en la exactitud obtenida, y la hipótesis nula H_0 : No hay diferencia en la exactitud de los métodos.

Ya establecida la hipótesis se establece la probabilidad de confianza para la prueba, en este caso se uso una probabilidad de 95% de confianza, lo cual nos da un valor $\alpha = 0.05$, y se evalúan las distribuciones tanto por la parte positiva como la negativa se realiza una prueba a dos colas, lo cual usando la distribución normal estándar encontramos que el valor z calculado debe encontrarse en el rango $[-1.96, 1.96]$ para

afirmar la hipótesis nula, de lo contrario se afirma la hipótesis esperada.

Lo siguiente es el calculo y suma de los rangos, para el calculo de los rangos se juntan datos tanto del método AE como los del AGC y se ordenan de mayor a menor y se le asigna un rango iniciando con el dato mas bajo se le asigna el rango uno, al segundo un dos y así hasta el ultimo. si existen datos que tengan el mismo valor, se calcula la media de los rangos y se les asigna la media como rango para todos, por ejemplo, si existen cuatro valores iguales y los rangos para estos datos corresponden a 4,5,6,7 se calcula la media $(4 + 5 + 6 + 7)/4 = 5.5$, y este es usado como el rango para los cuatro valores.

Ya que los los datos obtenidos son de igual tamaño no habría problema en usar los rangos de AE o de AGC, pero ya que el método AGC es el que estamos comparando contra el AE, se usaran los datos obtenidos usando AGC. Se calcula R que es la suma de los rangos para el método AGC.

El calculo del valor z esta dado por la ecuación (ec. 36) donde μ_R y σ_R se calculan con las ecuaciones (ec. 37), (ec. 38), respectivamente, donde n_1 y n_2 es la cantidad de datos usadas en cada método (treinta para ambos casos).

$$z = \frac{R - \mu_R}{\sigma_R} \quad (36)$$

$$\mu_R = \frac{n_1(n_1 + n_2 + 1)}{2} \quad (37)$$

$$\sigma_R = \sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}} \quad (38)$$

Ya que se usaron treinta casos para AE y otros treinta para AGC $n_1 = n_2 = 30$.

$$\mu_r = 915, \sigma_R = 67.6387 \quad (39)$$

Ya que el método RANSAC varía los tiempos dependiendo de que tan cerca este de encontrar un modelo que se ajuste a la nube de puntos, se considero realizar una comparación de la media de los tiempos de ejecución.

El proceso que realiza el sistema se dividió en ocho etapas que se describen en la tabla 3.1. y Se midieron los tiempos a la vez que se obtuvieron los datos para la precisión, dando un conjunto de cien datos para cada modelo y método usado.

Se realizaron seis casos, tres modelos distintos usando los métodos AE y AGC, Las tablas de los tiempos para cada caso se puede consultar en el apéndice A.

La prueba para el [plano](#) de realizo colocando un libro (objeto plano) como se muestra en la figura 3.4.

Y la prueba para el [cilindro](#) se realizo colocando un termo cilíndrico como se muestra en la figura 3.5.

Tabla 3.1: Descripción de los tiempos del sistema

-
- T1: La obtención de los datos de sensor Kinect.
 - T2: El filtrado de densidad de puntos.
 - T3: La resta de nubes de puntos.
 - T4: El agrupamiento de los puntos por objeto.
 - T5: Método RANSAC para la esfera.
 - T6: Método RANSAC para el plano.
 - T7: Método RANSAC para el cilindro.
 - T8: Renderizado del resultado.
-

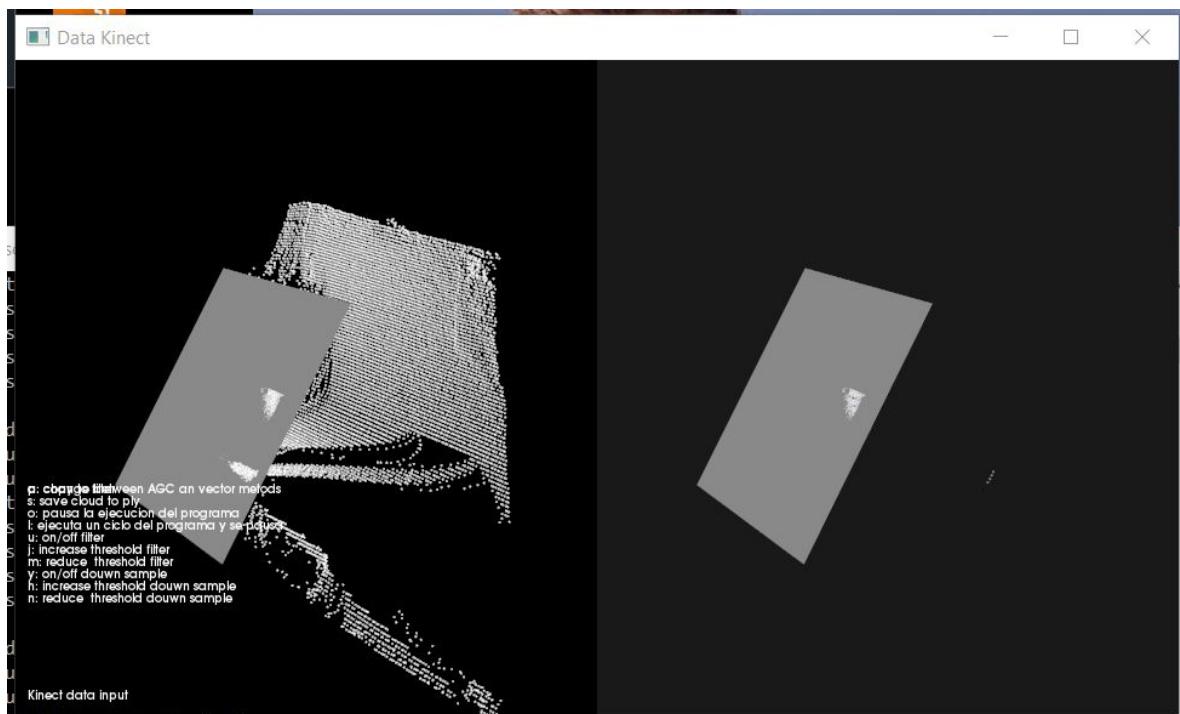


Figura 3.4: Prueba para el plano

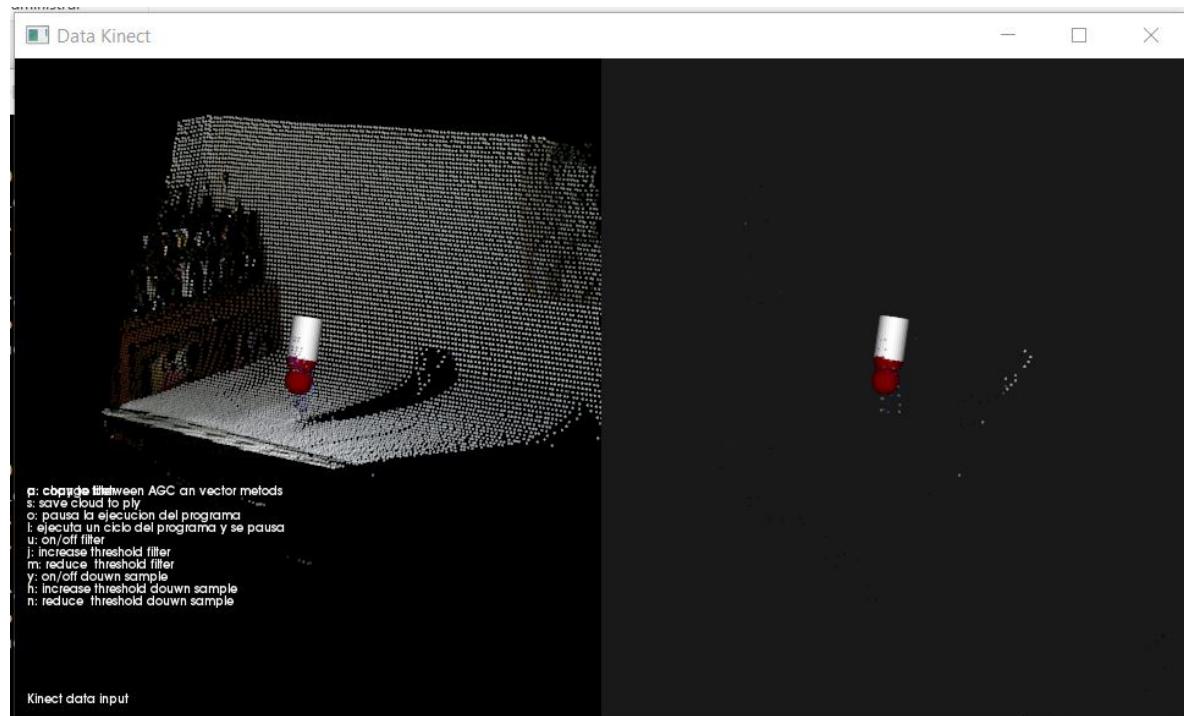


Figura 3.5: Prueba para el cilindro

4

ANÁLISIS DE RESULTADOS

Para analizar los datos obtenidos del caso de estudio presentado en el capítulo anterior, se visualizó la el caso de estudio de dos formas distintas; la primera una prueba de calidad usando la exactitud y precisión de cada método, y la segunda una prueba de rendimiento donde se comparó los tiempos de cada método.

Ya que se busca comparar el método usando AE y AGC para cada modelo, realizamos una comparación entre estos métodos para cada modelo propuesto.

4.1 OBJETO ESFÉRICO

Para comparar la calidad de la clasificación para representar una esfera, se calcula con la ecuación (ec. 34), ya que los datos se trabajaron en el lenguaje de programación R, para el calculo de la precisión se puede realizar usando el comando *mean()*. La precisión calculada para el método usando AE fue de 1.00, y usando AGC se obtuvo 0.99, aplicando un criterio en el cual consideramos que si los valores se mantienen en un intervalo de ± 0.025 se consideran como iguales, ya que $-0.025 < (1.00 - 0.99) < 0.025$, se obtiene que la precisión entre los métodos no varia.

Para la exactitud se evalúa usando la prueba de suma de rangos de Wilcoxon para datos independientes. A continuación se describen los pasos usados en el calculo de la prueba de Wilcoxon para la esfera.

En el código 4.1, para el lenguaje R, se muestra la lectura de los archivos de resultados y se carga en la variable *esferaAE* y *esferaAGC*, estas son tablas que contienen los tiempos parciales de ejecución, el *veredicto* es una columna con modelo elegido para representar al objeto, y el *metodo* se refiere al método usado, 0 para AE, y 1 para AGC. Se eliminan los casos que no aportan información a la prueba, esto es si no existe un veredicto, lo cual ocurre al inicio cuando aun no se captura la nube de puntos que sera el escenario, así como los que no representaron realizaron una buena representación, para el caso de la esfera el como del *veredicto* debe contener la palabra "sphere," ("plane," para el plano y "cylinder," para el cilindro), y por ultimo

también se quitan los casos cuando se usa un método distinto al deseado.

Código 4.1: Lectura de datos en R.

```

1 esferaAE <- read.csv("esferaAE.txt", sep="\t")
2 paraEliminar <- esferaAE$veredicto == "" |
3   esferaAE$veredicto == " , " | esferaAE$metodo != "o, "
4   esferaAE$veredicto != "sphere, "
5 esferaAE <- esferaAE[!paraEliminar,]
6
7 esferaAGC <- read.csv("esferaAGC.txt", sep="\t")
8 paraEliminar <- esferaAGC$veredicto == "" |
9   esferaAGC$veredicto == " , " | esferaAGC$metodo != "1, "
10  esferaAGC$veredicto != "sphere"
11 esferaAGC <- esferaAGC[!paraEliminar,]
```

A continuación en el código 4.1 se muestra la seleccionan los primeros treinta datos, de cada método, se juntan los datos en *esferaRank*, y se calcula el rango según el valor obtenido de exactitud, la exactitud fue calculada usando la ecuación (ec. 35).

Código 4.2: Asignación de rangos.

```

1 esferaAE30 <- esferaAE[1:30,]
2 esferaAGC30 <- esferaAGC[1:30,]
3
4 esferaRank<-rbind(esferaAE30,esferaAGC30)
5 esferaRank$rango=rank(esferaRank$exactitud)
```

En el código 4.1, se observa como se separa el grupo que usa AGC, y se realiza la suma de los rangos y se guarda en la variable *sum* (el numero 13 corresponde al indice de la columna de los rangos), se calcula μ_r , σ_R y z .

Código 4.3: Calculo de z.

```

1 paraEliminar <- esferaRank$metodo != "1, "
2 esferaRank<-esferaRank[!paraEliminar,]
3 sum<-colSums(esferaRank[,13, drop=FALSE])
4
5 mu=30*(30+30+1)/2
6 sigma=((30*30*(30+30+1))/12)^(1/2)
7 ##### se calcula z
```

8 $Z=(\text{sum}-\mu)/\sigma$

Obteniendo un valor $z = -5.6$, comparando z en el intervalo $[-1.96, 1.96]$ se observa que $z < -1.96$ por lo cual la hipótesis nula es descartada, y se afirma que la exactitud obtenida por el método usando AE es diferente a la obtenida por AGC. La media para el método usando AE es 0.9597 y usando AGC es 0.8581, con lo cual se asegura que para representar objetos esféricos el método usando AE es mas exacto que usando AGC.

4.2 OBJETO PLANO

De manera similar a la esfera se calcularon los valores de la exactitud y precisión para un modelo [plano](#).

Para evaluar la precisión al representar un objeto plano, el valor obtenido usando el método con AE es de 0.90 y usando AGC es de 0.94, aplicando el mismo criterio para el intervalo ± 0.025 , se obtiene que $(0.90 - 0.94) < -0.025$, con lo cual se afirma que la precisión entre los métodos es diferente, y que el método que usa AGC es mas preciso que el que usa AE.

Mientras que para la evaluación de la exactitud, se calculo un valor $z = -0.887$, ya que $-1.95 < -0.887 < 1.95$ se confirma la hipótesis nula, la cual afirma que la exactitud de los métodos para representar objetos planos es igual.

4.3 OBJETO CILÍNDRICO

Por último tenemos el modelo del [cilindro](#). Al igual que para el plano y la esfera se realizaron los cálculos de forma similar.

Para evaluar la precisión al representar un objeto cilíndrico, el valor obtenido usando el método con AE es de 0.47 y usando AGC es de 0.96, aplicando el mismo criterio para el intervalo ± 0.025 , se obtiene que $(0.47 - 0.96) < -0.025$, con lo cual se afirma que la precisión entre los métodos es diferente, y que el método que usa AGC es mas preciso que el que usa AE.

Mientras que para la evaluación de la exactitud, se calculo un valor $z = 0.761$, ya que $-1.95 < 0.761 < 1.95$ se confirma la hipótesis nula, la cual afirma que la exactitud de los métodos para representar objetos cilíndricos es igual.

4.4 RENDIMIENTO

Para los tiempos de ejecución, en la tabla 4.1 se muestran los promedios de los tiempos obtenidos para cada etapa del programa descritos anteriormente en la tabla 3.1. las filas describen el modelo del caso de estudio y el método usado. Para un mejor entendimiento la figura 4.1 muestra gráficamente los datos obtenidos.

Tabla 4.1: Resumen de tiempos de la prueba para la esfera usando AE

	T1(seg.)	T2(seg.)	T3(seg.)	T4(seg.)	T5(seg.)	T6(seg.)	T7(seg.)	T8(seg.)	T total(seg.)
Esfera AE	1.3122	0.2956	0.3712	0.0434	0.0088	0.2048	0.2769	0.0142	2.5284
Esfera AGC	1.0987	0.2938	0.3985	0.0465	0.0172	0.2184	0.0962	0.0217	2.1921
Plano AE	1.0745	0.2671	0.3414	0.0305	0.0128	0.0057	0.1741	0.0093	1.9165
Plano AGC	1.0974	0.2716	0.3555	0.0312	0.1715	0.0047	1.0286	0.0190	2.9803
Cilindro AE	0.8806	0.2962	0.3821	0.0245	0.0247	0.0137	0.0831	0.0146	1.7205
Cilindro AGC	0.9730	0.2947	0.3775	0.0231	0.0447	0.0135	0.2322	0.0404	1.9997

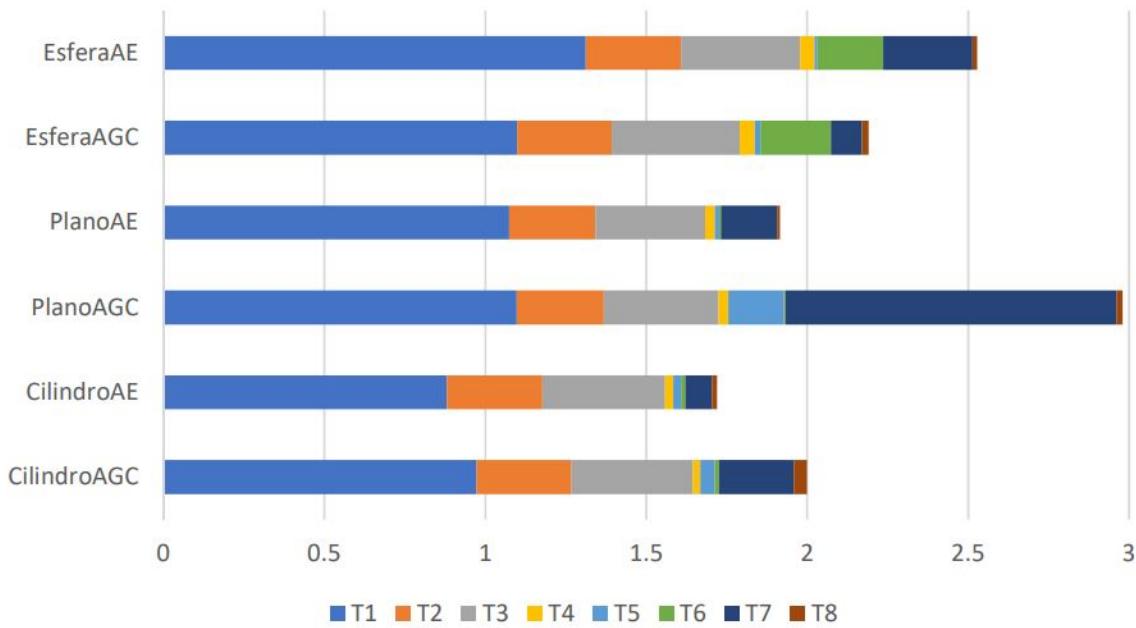


Figura 4.1: Tiempos promedios de la prueba

5

CONSIDERACIONES FINALES

5.1 CONCLUSIONES

La precisión en el proceso de clasificado al usar AGC fue mayor en cilindros y planos, pero no hubo diferencias en la esfera. Al evaluar la exactitud los valores para el cilindro y el plano no presentaron cambios y en la esfera disminuyo cuando se usó AGC para clasificar los objetos. En conclusión, se recomienda el uso de RANSAC modificado con AGC para las geometrías cilíndricas, pero no se encontró mejoras para planos y no se recomienda para esferas.

En este trabajo se planteó el desarrollo de un sistema con la capacidad de representar objetos usando figuras geométricas usando AGC, a partir de un problema de clasificación.

Se investigó y desarrolló un sistema que se comunica con el sensor Kinect y acomoda los datos obtenidos en una nube de puntos.

Se investigaron las técnicas de visión por computadora de la transformada de Hough y redes neuronales como posibles clasificadores, cada uno con sus propias limitaciones dando lugar al algoritmo RANSAC como el método que más se ajusta a las necesidades del sistema.

Se desarrolló un sistema que permite representar objetos reales según su geometría y presentarlos en un ambiente virtual usando RANSAC para obtener las geometrías.

Se modificó el algoritmo RANSAC para usarlo con AGC el cual comparado con el algoritmo usando AE de la biblioteca PCL se obtuvieron los resultados desfavorables en la exactitud para la detección de esferas, y con precisión similar entre AE y AGC para las otras geometrías, pero con resultados favorables para la precisión con objetos planos y cilíndricos, y precisión similar para objetos esféricos.

5.2 TRABAJO A FUTURO

Como posibles trabajos a futuro se tiene:

- Realización de una pruebas de rendimiento.

- Paralelización del sistema, modificar la forma de trabajo para realizar la obtención de los modelos de forma paralela, cada que se obtienen nuevos datos del sensor generar un hilo para obtener resultados mas fluidos, así como la paralelización de operaciones en AGC,
- Implementación de métodos RANSAC modificados como PROSAC (PROgressive SAmple Consensus), Preemptive RANSAC o R-RANSAC, cada método mejora cierto aspecto del algoritmo, mejores estimaciones, más rápidos, etc.
- Modificar el sistema para usar una combinación de los dos métodos y aprovechar las mejores características de cada uno, así beneficiarse de los tiempos obtenidos en modelos AE y la precisión obtenida para AGC.
- Mejorar el modelo obtenido aplicando el método de mínimos cuadrados.
- Desarrollo de modelos que describan mas geometrías (prismas, pirámides, etc.).

REFERENCIAS

- [1] "Motion capture systems — vicon," <https://www.vicon.com/>, (Accessed on 12/10/2018).
- [2] G. Martinsanz, G. PAJARES, and J. de la Cruz García, *Ejercicios Resueltos de Visión por Computador*. RA-MA S.A. Editorial y Publicaciones, 2007.
- [3] M. Weinmann, *Reconstruction and Analysis of 3D Scenes: From Irregularly Distributed 3D Points to Object Classes*. Springer International Publishing, 2016.
- [4] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.
- [5] M. A. i Melchor, "Uso del kinect en el computador desde el punto de vista del usuario," <https://riunet.upv.es/bitstream/handle/10251/83409/Agust%C3%ADn%20Uso%2odel%20Kinect%2oen%2oel%2ocomputador%2odesde%2oel%2opunto%2odel%2ovista%2odel%2ousuario.pdf?sequence=1>.
- [6] "Microsoft kills kinect support in xbox one's backward-compatibility push — pcworld," <https://www.pcworld.com/article/2936263/microsoft-kills-kinect-support-in-xbox-ones-backward-compatibility-push.html>, (Accessed on 12/10/2018).
- [7] E. Lachat, H. Macher, T. Landes, and P. Grussenmeyer, "Assessment and calibration of a RGB-d camera (kinect v2 sensor) towards a potential use for close-range 3d modeling," *Remote Sensing*, vol. 7, no. 10, pp. 13 070–13 097, oct 2015.
- [8] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer Science and Business Media, 2010.
- [9] F. Sanguino, T. J. Mateo; Ponce Gómez, "Improving 3d object detection and classification based on kinect sensor and hough transform," in *2015 International Symposium on Innovations in Intelligent SysTems and Applications (INISTA)*, 2015.
- [10] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, jun 1981. [Online]. Available: <https://doi.org/10.1145/358669.358692>
- [11] D. Hildenbrand, *Foundations of Geometric Algebra Computing*. Springer Berlin Heidelberg, 2013.
- [12] I. Herstein, *Algebra abstracta*. Grupo Editorial Iberoamérica, 1988.

- [13] S. Grossman, *Álgebra lineal*, ser. Elibro Catedra. McGraw-Hill Interamericana de España S.L., 2008.
- [14] E. Bayro-Corrochano, *Geometric Computing: for Wavelet Transforms, Robot Vision, Learning, Control and Action*. Springer London, 2010.
- [15] A. Lasenby, "Recent applications of conformal geometric algebra," in *Computer Algebra and Geometric Algebra with Applications*, Berlin, Heidelberg, 2005.
- [16] E. B. C. E. Garza Burgos, M.; Sanchez O., "Medical robot vision using the conformal geometric algebra framework," in *Humanoid Robots (Humanoids)*, Cancun, Mexico, 2016.
- [17] S. D. X. J. S. Jungong, Han; Ling, "Enhanced computer vision with microsoft kinect sensor: A review," *IEEE Transactions on Cybernetics*, vol. 43, no. 5, 2013.
- [18] J. N. Simón, "Reconocimiento de objetos mediante sensor 3d," Master's thesis, Universidad Carlos III de Madrid, Madrid, 2012.
- [19] A. Sveier, A. L. Kleppe, L. Tingelstad, and O. Egeland, "Object detection in point clouds using conformal geometric algebra," *Advances in Applied Clifford Algebras*, vol. 27, no. 3, pp. 1961–1976, feb 2017.
- [20] "ViveTM — discover virtual reality beyond imagination," <https://www.vive.com/us/>, (Accessed on 11/30/2018).
- [21] "Oculus rift: Visor de realidad virtual para pc optimizadas para realidad virtual — oculus," <https://www.oculus.com/rift/#oui-csl-rift-games=robo-recall>, (Accessed on 11/30/2018).
- [22] "Microsoft hololens — the leader in mixed reality technology," <https://www.microsoft.com/en-us/hololens>, (Accessed on 12/10/2018).
- [23] "Xbox one kinect sensor para xone - gameplanet," <https://gameplanet.com/xbox-one-kinect-sensor-xone.html>, (Accessed on 12/10/2018).
- [24] Lingzhu Xiang, F. Echtler, C. Kerl, and et. al., "Libfreenect2: Release 0.2," 2016. [Online]. Available: <https://zenodo.org/record/50641>
- [25] "giacomodabisias/libfreenect2pclgrabber: a grabber for the kinect2 which returns a registered pcl point cloud," <https://github.com/giacomodabisias/libfreenect2pclgrabber>, (Accessed on 12/10/2018).
- [26] "Cinderella : Cinderella," <https://www.cinderella.de/tiki-index.php>, (Accessed on 12/10/2018).
- [27] "Geometric algebra for computer science," <http://www.geometricalgebra.net/index.html>, (Accessed on 12/10/2018).
- [28] "Clucalc," <http://www.clucalc.info/>, (Accessed on 12/10/2018).

- [29] "Gaalop — geometric algebra algorithms optimizer," <http://www.gaalop.de/>, (Accessed on 12/10/2018).
- [30] "versor.mat.ucsb.edu," <http://versor.mat.ucsb.edu/>, (Accessed on 12/10/2018).
- [31] "weshoke/versor.js: A javascript port of the versor geometric algebra library," <https://github.com/weshoke/versor.js>, (Accessed on 12/10/2018).
- [32] A. Bluman, *Elementary Statistics: A Step by Step Approach*. McGraw-Hill Higher Education, 2009. [Online]. Available: <https://books.google.com.mx/books?id=TyLcPgAACAAJ>

Apéndices

A PRUEBAS

Se presentan los datos obtenidos durante las pruebas para el caso de estudio, los datos se separan por tabuladores en doce datos:

- La obtención de los datos de sensor Kinect (segundos).
- El filtrado de densidad de puntos(segundos).
- La resta de nubes de puntos(segundos).
- El agrupamiento de los puntos por objeto(segundos).
- Método RANSAC para la esfera(segundos).
- Método RANSAC para el plano(segundos).
- Método RANSAC para el cilindro(segundos).
- Clasificaron del objeto.
- Exactitud del modelo.
- Renderizado del resultado(segundos).
- Tiempo total de ejecución(segundos).

Código A.1: Datos obtenidos para la prueba del plano usando AE.

1	0.598	0.244	0.339	0.035	0.011	0.005	0.074	plane ,	0.841270	0.007	0,	1.313
2	0.622	0.245	0.318	0.022	0.02	0.003	0.207	plane ,	0.941176	0.006	0,	1.445
3	0.673	0.244	0.31	0.031	0.01	0.005	0.212	plane ,	0.947826	0.005	0,	1.492
4	0.597	0.249	0.344	0.017	0.008	0.002	0.165	plane ,	0.939655	0.007	0,	1.39
5	0.618	0.242	0.303	0.017	0.011	0.002	0.155	plane ,	0.973684	0.019	0,	1.367
6	0.689	0.253	0.331	0.03	0.011	0.012	0.138	plane ,	0.947368	0.006	0,	1.47
7	0.638	0.259	0.322	0.032	0.012	0.006	0.164	plane ,	0.981818	0.015	0,	1.449
8	0.676	0.276	0.328	0.033	0.019	0.007	0.15	plane ,	0.926606	0.015	0,	1.504
9	0.637	0.262	0.34	0.025	0.014	0.005	0.196	cylinder ,	0.932203	0.01	0,	1.489
10	0.701	0.259	0.34	0.033	0.008	0.003	0.249	plane ,	0.991525	0.017	0,	1.611
11	0.619	0.276	0.362	0.037	0.004	0.003	0.139	plane ,	1.000000	0.006	0,	1.448
12	0.626	0.269	0.338	0.023	0.008	0.003	0.155	plane ,	0.990826	0.01	0,	1.435
13	0.71	0.285	0.333	0.033	0.009	0.003	0.189	plane ,	0.938596	0.005	0,	1.568
14	0.761	0.266	0.346	0.034	0.014	0.004	0.123	cylinder ,	1.000000	0.013	0,	1.562
15	0.615	0.264	0.341	0.035	0.015	0.004	0.241	plane ,	0.956522	0.008	0,	1.524

16	0.874	0.258	0.322	0.021	0.018	0.005	0.143	plane ,	0.911504	0.008	0,	1.651
17	0.658	0.253	0.367	0.033	0.037	0.005	0.15	cylinder ,	0.939130	0.011	0,	1.516
18	0.624	0.267	0.334	0.025	0.011	0.004	0.156	plane ,	0.974138	0.007	0,	1.43
19	0.701	0.293	0.348	0.024	0.008	0.026	0.175	plane ,	0.940678	0.017	0,	1.593
20	0.637	0.27	0.342	0.039	0.015	0.004	0.157	plane ,	0.972727	0.008	0,	1.472
21	0.689	0.26	0.349	0.035	0.022	0.011	0.219	plane ,	0.964912	0.007	0,	1.592
22	0.634	0.268	0.33	0.023	0.013	0.005	0.18	cylinder ,	0.956522	0.013	0,	1.468
23	0.723	0.271	0.327	0.032	0.012	0.003	0.191	plane ,	0.965812	0.019	0,	1.578
24	0.649	0.274	0.343	0.016	0.01	0.004	0.149	plane ,	0.972973	0.008	0,	1.454
25	0.643	0.268	0.35	0.034	0.018	0.004	0.222	plane ,	0.956140	0.005	0,	1.546
26	0.682	0.273	0.338	0.034	0.008	0.003	0.178	plane ,	0.965517	0.007	0,	1.523
27	0.676	0.276	0.349	0.032	0.016	0.003	0.221	plane ,	0.947826	0.006	0,	1.58
28	0.616	0.274	0.35	0.024	0.024	0.003	0.166	plane ,	0.974359	0.006	0,	1.464
29	0.713	0.269	0.338	0.034	0.015	0.004	0.15	plane ,	0.947368	0.007	0,	1.53
30	0.659	0.255	0.334	0.024	0.006	0.005	0.164	plane ,	0.981818	0.009	0,	1.456
31	0.678	0.268	0.348	0.03	0.004	0.005	0.189	plane ,	0.940678	0.007	0,	1.529
32	0.64	0.271	0.333	0.024	0.024	0.005	0.209	plane ,	0.972222	0.008	0,	1.514
33	0.728	0.265	0.356	0.031	0.019	0.003	0.245	plane ,	0.965517	0.005	0,	1.652
34	0.656	0.269	0.335	0.023	0.015	0.003	0.175	plane ,	0.971963	0.008	0,	1.484
35	0.665	0.258	0.351	0.021	0.005	0.015	0.169	plane ,	0.965217	0.007	0,	1.493
36	0.718	0.27	0.344	0.025	0.011	0.015	0.176	plane ,	0.954545	0.006	0,	1.565
37	0.604	0.254	0.367	0.038	0.012	0.004	0.17	plane ,	0.973214	0.006	0,	1.455
38	0.661	0.272	0.336	0.034	0.013	0.003	0.238	plane ,	0.941176	0.008	0,	1.566
39	0.707	0.269	0.341	0.024	0.022	0.005	0.141	plane ,	0.981982	0.007	0,	1.516
40	0.604	0.266	0.342	0.041	0.01	0.005	0.184	plane ,	1.000000	0.007	0,	1.459
41	0.656	0.295	0.347	0.032	0.026	0.004	0.284	plane ,	0.941176	0.046	0,	1.69
42	0.75	0.278	0.357	0.037	0.005	0.004	0.186	plane ,	0.972477	0.01	0,	1.627
43	0.699	0.265	0.337	0.024	0.009	0.003	0.169	plane ,	0.973214	0.006	0,	1.513
44	0.687	0.287	0.348	0.038	0.011	0.003	0.173	plane ,	0.958678	0.011	0,	1.558
45	0.82	0.284	0.35	0.047	0.01	0.003	0.14	plane ,	0.981982	0.011	0,	1.665
46	0.669	0.257	0.353	0.035	0.013	0.006	0.352	plane ,	0.939055	0.008	0,	1.694
47	0.763	0.476	0.487	0.025	0.01	0.009	0.17	plane ,	0.981481	0.007	0,	1.948
48	0.779	0.357	0.538	0.027	0.014	0.005	0.243	plane ,	0.990909	0.01	0,	1.974
49	0.79	0.263	0.374	0.023	0.021	0.003	0.123	plane ,	1.000000	0.006	0,	1.604
50	0.748	0.288	0.459	0.043	0.007	0.077	0.229	plane ,	0.974359	0.014	0,	1.865
51	1.447	0.318	0.375	0.033	0.015	0.004	0.238	plane ,	0.956522	0.008	0,	2.441
52	1.321	0.288	0.384	0.036	0.008	0.003	0.158	plane ,	0.965812	0.011	0,	2.211
53	1.081	0.278	0.405	0.153	0.057	0.003	0.196	plane ,	0.973451	0.009	0,	2.183
54	0.529	0.226	0.311	0.034	0.038	0.003	0.244	plane ,	0.990991	0.011	0,	1.398
55	1.765	0.232	0.328	0.026	0.022	0.003	0.114	plane ,	0.981982	0.006	0,	2.497
56	0.741	0.339	0.31	0.031	0.008	0.003	0.165	plane ,	0.963964	0.007	0,	1.605
57	3.464	0.227	0.308	0.019	0.016	0.003	0.13	plane ,	0.974790	0.006	0,	4.173
58	0.615	0.231	0.307	0.026	0.009	0.003	0.145	plane ,	0.990991	0.007	0,	1.343
59	6.528	0.216	0.315	0.03	0.01	0.003	0.146	plane ,	0.911504	0.007	0,	7.255
60	1.501	0.227	0.293	0.03	0.009	0.003	0.147	plane ,	0.981818	0.014	0,	2.225
61	3.201	0.218	0.285	0.027	0.011	0.002	0.155	plane ,	0.972727	0.019	0,	3.919
62	2.732	0.226	0.337	0.029	0.006	0.004	0.156	plane ,	0.947368	0.016	0,	3.509
63	1.078	0.278	0.300	0.03	0.02	0.012	0.149	plane ,	0.983051	0.007	0,	1.874
64	2.279	0.248	0.276	0.019	0.013	0.008	0.141	plane ,	0.923077	0.006	0,	2.991
65	2.252	0.235	0.304	0.021	0.015	0.004	0.141	plane ,	0.931034	0.007	0,	2.979
66	0.545	0.771	0.33	0.029	0.01	0.004	0.132	plane ,	0.946429	0.006	0,	1.827
67	0.556	0.237	0.291	0.03	0.01	0.003	0.162	plane ,	0.973214	0.006	0,	1.295
68	3.162	0.276	0.426	0.035	0.007	0.006	0.164	cylinder ,	0.905983	0.01	0,	4.087
69	0.694	0.234	0.31	0.019	0.016	0.003	0.192	plane ,	0.956522	0.005	0,	1.474
70	0.813	0.273	0.312	0.031	0.005	0.006	0.163	plane ,	0.981818	0.007	0,	1.612
71	0.568	0.266	0.448	0.019	0.024	0.004	0.161	cylinder ,	0.955357	0.011	0,	1.501
72	0.515	0.239	0.337	0.03	0.004	0.003	0.159	plane ,	0.981818	0.016	0,	1.303
73	2.262	0.234	0.307	0.016	0.007	0.004	0.12	plane ,	0.963964	0.015	0,	2.976
74	0.521	0.237	0.293	0.032	0.01	0.005	0.174	plane ,	0.931034	0.006	0,	1.278
75	2.055	0.219	0.315	0.024	0.01	0.003	0.148	plane ,	0.939130	0.006	0,	2.781
76	0.544	0.221	0.263	0.034	0.019	0.004	0.216	plane ,	0.941176	0.017	0,	1.321
77	1.369	0.23	0.308	0.03	0.006	0.003	0.177	cylinder ,	0.947826	0.008	0,	2.132
78	0.533	0.25	0.43	0.026	0.006	0.004	0.199	plane ,	0.947826	0.006	0,	1.456
79	2.311	0.225	0.36	0.021	0.018	0.003	0.198	plane ,	0.962617	0.016	0,	3.152

80	0.886	0.248	0.324	0.029	0.011	0.003	0.136	cylinder ,	0.964286	0.008	0,	1.646
81	0.67	0.249	0.332	0.04	0.009	0.006	0.187	plane ,	0.899083	0.016	0,	1.509
82	0.525	0.619	0.334	0.027	0.008	0.005	0.168	plane ,	0.972477	0.007	0,	1.694
83	1.609	0.211	0.301	0.029	0.008	0.005	0.179	plane ,	0.963303	0.007	0,	2.351
84	0.541	0.211	0.309	0.03	0.006	0.004	0.153	plane ,	0.914530	0.006	0,	1.261
85	0.523	0.229	0.466	0.027	0.005	0.003	0.173	plane ,	0.931034	0.006	0,	1.433
86	1.923	0.24	0.281	0.022	0.019	0.004	0.183	cylinder ,	0.94958	0.01	0,	2.682
87	0.542	0.255	0.402	0.034	0.007	0.005	0.131	plane ,	0.972973	0.006	0,	1.383
88	5.157	0.231	0.339	0.03	0.01	0.003	0.118	plane ,	0.973214	0.007	0,	5.895
89	0.741	0.268	0.358	0.025	0.018	0.003	0.193	plane ,	0.928571	0.005	0,	1.613
90	2.297	0.242	0.303	0.033	0.009	0.003	0.147	plane ,	0.991071	0.005	0,	3.04
91	0.525	0.238	0.335	0.03	0.005	0.014	0.134	plane ,	0.948718	0.019	0,	1.301
92	1.735	0.256	0.407	0.038	0.014	0.004	0.206	plane ,	1.000000	0.008	0,	2.67
93	0.725	0.224	0.343	0.036	0.004	0.003	0.159	plane ,	0.981818	0.007	0,	1.501
94	0.714	0.269	0.374	0.025	0.006	0.014	0.233	plane ,	0.917431	0.006	0,	1.643
95	2.972	0.22	0.271	0.017	0.01	0.004	0.121	plane ,	0.981982	0.005	0,	3.621
96	0.577	0.248	0.33	0.017	0.008	0.013	0.125	plane ,	0.972973	0.006	0,	1.326
97	1.178	0.254	0.282	0.031	0.013	0.013	0.161	plane ,	0.852174	0.007	0,	1.941
98	0.508	0.217	0.297	0.035	0.009	0.002	0.184	plane ,	0.983333	0.007	0,	1.267
99	1.641	0.223	0.317	0.027	0.01	0.003	0.132	cylinder ,	1.000000	0.009	0,	2.364
100	0.495	0.229	0.351	0.058	0.018	0.004	0.258	plane ,	0.981982	0.009	0,	1.422

Código A.2: Datos obtenidos para la prueba del plano usando AGC.

1	0.574	0.239	0.304	0.032	0.531	0.015	2.937	plane	0.175879	0.012	1,	4.644
2	0.874	0.342	0.416	0.026	0.269	0.004	1.011	plane	0.803419	0.024	1,	2.966
3	0.799	0.34	0.374	0.032	0.003	0.005	1.215	plane	0.983193	0.012	1,	2.78
4	0.828	0.294	0.362	0.041	0.261	0.004	1.203	plane	0.974576	0.012	1,	3.007
5	0.83	0.286	0.365	0.033	0.265	0.004	1.144	plane	0.965812	0.025	1,	2.953
6	0.966	0.27	0.346	0.028	0.261	0.004	0.795	plane	0.909091	0.028	1,	2.698
7	0.838	0.274	0.345	0.034	0.003	0.004	1.006	plane	0.949580	0.026	1,	2.531
8	0.665	0.269	0.373	0.111	0.255	0.006	1.532	plane	0.559322	0.021	1,	3.234
9	1.471	0.299	0.346	0.041	0.21	0.004	0.763	plane	0.982906	0.017	1,	3.151
10	0.815	0.283	0.337	0.024	0.22	0.004	0.903	plane	0.434783	0.012	1,	2.598
11	1.331	0.282	0.346	0.021	0.002	0.004	0.83	plane	0.974359	0.015	1,	2.832
12	0.771	0.426	0.359	0.035	0.241	0.003	1.17	plane	0.957983	0.014	1,	3.02
13	0.812	0.356	0.354	0.043	0.337	0.016	0.701	plane	0.878261	0.012	1,	2.632
14	1.123	0.271	0.401	0.047	0.003	0.003	1.295	plane	0.897436	0.014	1,	3.158
15	0.688	0.268	0.344	0.032	0.221	0.004	0.651	plane	0.973913	0.012	1,	2.221
16	10.201	0.27	0.342	0.017	0.213	0.006	1.293	plane	0.286957	0.012	1,	12.354
17	1.913	0.269	0.347	0.025	0.231	0.005	0.962	plane	0.933333	0.013	1,	3.765
18	3.202	0.282	0.347	0.032	0.002	0.006	0.949	plane	0.966102	0.014	1,	4.835
19	12.494	0.257	0.341	0.019	0.235	0.005	0.445	plane	0.933884	0.014	1,	13.81
20	4.167	0.248	0.324	0.033	0.241	0.005	0.732	plane	0.881356	0.02	1,	5.77
21	0.839	0.292	0.33	0.034	0.215	0.003	1.14	plane	0.991453	0.025	1,	2.879
22	0.693	0.261	0.342	0.025	0.173	0.004	0.816	plane	0.862069	0.014	1,	2.328
23	0.800	0.319	0.341	0.037	0.002	0.004	1.102	plane	0.991379	0.013	1,	2.619
24	0.673	0.342	0.556	0.025	0.224	0.003	1.615	plane	0.864407	0.041	1,	3.48
25	1.411	0.283	0.383	0.026	0.233	0.003	1.021	plane	0.957265	0.011	1,	3.375
26	0.791	0.279	0.347	0.033	0.231	0.005	1.568	plane	0.975410	0.011	1,	3.265
27	0.751	0.283	0.339	0.034	0.004	0.004	1.209	plane	0.966667	0.295	1,	2.92
28	0.799	0.28	0.342	0.023	0.006	0.007	0.014	plane	0.983193	0.014	1,	1.485
29	0.64	0.277	0.353	0.024	0.253	0.005	1.137	plane	0.975000	0.014	1,	2.706
30	0.709	0.277	0.347	0.023	0.234	0.012	1.374	plane	0.975000	0.023	1,	3.002
31	0.652	0.256	0.347	0.026	0.002	0.003	1.161	plane	0.411765	0.028	1,	2.475
32	0.65	0.368	0.394	0.033	0.226	0.003	0.895	plane	1.000000	0.025	1,	2.595
33	0.717	0.29	0.341	0.03	0.217	0.004	0.666	plane	0.973684	0.011	1,	2.277
34	0.634	0.28	0.333	0.025	0.23	0.005	0.916	plane	0.974359	0.025	1,	2.448
35	0.593	0.264	0.442	0.037	0.284	0.004	2.225	plane	0.983333	0.024	1,	3.873
36	0.742	0.281	0.433	0.057	0.027	0.008	0.945	plane	0.949153	0.013	1,	2.508

37	0.747	0.275	0.38	0.021	0.003	0.003	0.811	plane	0.966667	0.012	1,	2.252
38	0.673	0.339	0.458	0.038	0.214	0.004	0.934	plane	0.933884	0.019	1,	2.686
39	0.797	0.469	0.352	0.035	0.24	0.006	1.336	plane	0.941667	0.014	1,	3.25
40	0.701	0.288	0.385	0.033	0.252	0.003	1.435	sphere	0.260504	0.021	1,	3.118
41	0.771	0.269	0.369	0.024	0.242	0.004	0.892	plane	0.966667	0.012	1,	2.584
42	0.712	0.254	0.371	0.034	0.248	0.003	0.977	plane	0.933333	0.024	1,	2.623
43	0.615	0.251	0.385	0.032	0.003	0.004	1.338	plane	0.982759	0.014	1,	2.642
44	0.716	0.269	0.355	0.038	0.002	0.004	1.084	plane	0.504202	0.023	1,	2.491
45	0.671	0.257	0.378	0.024	0.21	0.004	1.201	sphere	0.230769	0.014	1,	2.759
46	0.686	0.282	0.386	0.032	0.002	0.003	0.928	plane	0.291667	0.025	1,	2.344
47	0.631	0.269	0.352	0.025	0.217	0.004	1.321	plane	0.973913	0.011	1,	2.83
48	0.701	0.24	0.391	0.023	0.236	0.005	0.68	plane	0.950413	0.025	1,	2.302
49	0.617	0.276	0.351	0.034	0.23	0.005	1.198	plane	0.922414	0.014	1,	2.726
50	0.738	0.257	0.348	0.032	0.002	0.004	1.066	plane	0.817391	0.02	1,	2.468
51	0.662	0.263	0.368	0.037	0.24	0.004	0.897	plane	0.991304	0.024	1,	2.496
52	0.746	0.255	0.371	0.037	0.002	0.003	0.694	plane	0.975610	0.012	1,	2.121
53	0.615	0.27	0.372	0.023	0.242	0.004	1.205	plane	0.983471	0.023	1,	2.755
54	0.711	0.257	0.346	0.026	0.227	0.004	1.144	plane	0.966667	0.012	1,	2.727
55	0.671	0.266	0.361	0.023	0.003	0.004	0.913	plane	0.807018	0.011	1,	2.253
56	0.673	0.252	0.364	0.033	0.002	0.003	0.866	plane	0.129310	0.011	1,	2.204
57	0.600	0.268	0.359	0.027	0.239	0.003	1.355	plane	0.966387	0.024	1,	2.876
58	0.707	0.265	0.357	0.021	0.244	0.003	0.718	sphere	0.235772	0.026	1,	2.342
59	0.603	0.266	0.347	0.032	0.228	0.005	1.109	plane	0.940171	0.021	1,	2.612
60	0.742	0.269	0.38	0.038	0.003	0.004	1.246	plane	0.641667	0.013	1,	2.695
61	0.746	0.309	0.37	0.032	0.23	0.003	1.429	plane	0.949580	0.012	1,	3.132
62	0.731	0.287	0.375	0.026	0.235	0.004	1.172	plane	0.475410	0.014	1,	2.844
63	0.634	0.262	0.369	0.035	0.244	0.004	0.701	plane	0.991803	0.026	1,	2.276
64	0.744	0.262	0.38	0.039	0.261	0.005	1.165	plane	0.588235	0.025	1,	2.882
65	0.677	0.256	0.378	0.038	0.224	0.003	0.896	sphere	0.213675	0.017	1,	2.49
66	0.69	0.28	0.54	0.035	0.216	0.003	0.664	plane	0.974138	0.017	1,	2.446
67	0.653	0.263	0.363	0.034	0.231	0.004	1.54	sphere	0.245902	0.022	1,	3.11
68	1.185	0.276	0.347	0.028	0.149	0.003	1.267	plane	0.914530	0.013	1,	3.27
69	0.796	0.412	0.347	0.018	0.227	0.013	0.822	plane	0.631579	0.009	1,	2.645
70	0.504	0.231	0.293	0.03	0.284	0.005	0.825	plane	0.966667	0.011	1,	2.184
71	2.056	0.241	0.289	0.018	0.207	0.004	0.973	plane	0.491525	0.011	1,	3.799
72	0.489	0.206	0.312	0.022	0.229	0.003	1.209	sphere	0.225000	0.011	1,	2.482
73	0.527	0.238	0.274	0.035	0.002	0.005	1.228	plane	0.663793	0.023	1,	2.332
74	1.731	0.246	0.31	0.03	0.228	0.003	0.843	plane	0.906780	0.011	1,	3.402
75	0.65	0.241	0.365	0.03	0.194	0.003	1.001	plane	0.896552	0.021	1,	2.506
76	0.626	0.223	0.351	0.022	0.013	0.004	0.784	plane	0.000000	0.021	1,	2.044
77	1.716	0.211	0.301	0.028	0.184	0.003	0.81	plane	0.275862	0.013	1,	3.266
78	0.552	0.343	0.381	0.02	0.199	0.003	1.126	plane	0.957627	0.013	1,	2.638
79	0.623	0.24	0.358	0.025	0.004	0.012	0.663	plane	0.876033	0.02	1,	1.945
80	0.682	0.321	0.347	0.039	0.232	0.003	0.877	plane	0.966102	0.017	1,	2.519
81	1.465	0.253	0.303	0.031	0.205	0.004	0.591	plane	0.965812	0.013	1,	2.865
82	0.537	0.251	0.317	0.042	0.243	0.006	0.454	plane	0.932773	0.012	1,	1.863
83	0.606	0.256	0.33	0.029	0.213	0.011	0.422	plane	0.393162	0.012	1,	1.879
84	2.531	0.229	0.269	0.034	0.205	0.002	1.376	plane	0.991525	0.016	1,	4.663
85	0.603	0.226	0.33	0.029	0.211	0.003	1.029	plane	0.942623	0.011	1,	2.443
86	1.921	0.254	0.336	0.021	0.002	0.012	0.789	plane	0.939655	0.013	1,	3.348
87	0.547	0.219	0.281	0.036	0.003	0.005	0.842	plane	0.958333	0.01	1,	1.943
88	0.643	0.236	0.332	0.022	0.21	0.002	1.176	plane	0.974359	0.013	1,	2.635
89	1.855	0.211	0.265	0.016	0.245	0.004	1.009	plane	0.950820	0.012	1,	3.617
90	0.535	0.213	0.319	0.04	0.229	0.003	0.817	plane	0.420168	0.011	1,	2.167
91	0.542	0.221	0.307	0.041	0.224	0.004	1.03	plane	0.852459	0.012	1,	2.382
92	1.848	0.231	0.313	0.032	0.223	0.003	1.276	plane	0.848739	0.013	1,	3.939
93	0.555	0.202	0.312	0.038	0.003	0.004	1.15	plane	0.931624	0.009	1,	2.274
94	0.551	0.229	0.344	0.027	0.207	0.014	1.122	plane	0.932203	0.012	1,	2.506
95	0.653	0.245	0.394	0.032	0.001	0.003	0.841	plane	0.893443	0.02	1,	2.192
96	0.862	0.314	0.296	0.028	0.184	0.003	1.358	plane	1.000000	0.011	1,	3.057
97	0.531	0.200	0.321	0.029	0.231	0.003	0.602	plane	0.9743590	0.011	1,	1.928
98	0.721	0.236	0.472	0.024	0.21	0.003	0.79	plane	0.900000	0.011	1,	2.467
99	0.554	0.241	0.435	0.03	0.222	0.004	0.867	plane	0.983193	0.01	1,	2.363
100	0.791	0.251	0.325	0.029	0.002	0.006	0.815	plane	0.769231	0.01	1,	2.229

Código A.3: Datos obtenidos para la prueba de la esfera usando AE.

1	0.595	0.258	0.35	0.036	0.031	0.121	0.301	sphere ,	0.670103	0.008	0,	1.702
2	0.699	0.273	0.339	0.056	0.035	0.115	0.414	sphere ,	0.593886	0.01	0,	1.941
3	0.649	0.259	0.37	0.049	0.006	0.243	0.329	sphere ,	0.970588	0.021	0,	1.927
4	1.226	0.269	0.35	0.036	0.016	0.223	0.268	sphere ,	0.987952	0.011	0,	2.4
5	1.031	0.275	0.36	0.027	0.006	0.173	0.27	sphere ,	0.974684	0.018	0,	2.161
6	0.977	0.266	0.331	0.043	0.005	0.222	0.395	sphere ,	0.982036	0.046	0,	2.286
7	1.243	0.404	0.361	0.037	0.008	0.221	0.257	sphere ,	0.993976	0.02	0,	2.552
8	0.678	0.272	0.63	0.119	0.166	0.209	0.249	sphere ,	0.945455	0.013	0,	2.336
9	0.751	0.297	0.414	0.046	0.007	0.209	0.265	sphere ,	0.969325	0.012	0,	2.001
10	0.664	0.261	0.454	0.081	0.006	0.313	0.304	sphere ,	1.000000	0.011	0,	2.094
11	0.664	0.277	0.353	0.034	0.007	0.215	0.36	sphere ,	0.913580	0.009	0,	1.92
12	0.722	0.268	0.363	0.04	0.006	0.165	0.246	sphere ,	0.938272	0.02	0,	1.83
13	0.645	0.271	0.353	0.046	0.005	0.297	0.312	sphere ,	0.901235	0.008	0,	1.938
14	0.802	0.273	0.449	0.044	0.005	0.357	0.316	sphere ,	0.943396	0.02	0,	2.266
15	0.642	0.267	0.336	0.035	0.013	0.152	0.276	sphere ,	0.987654	0.009	0,	1.732
16	3.343	0.261	0.328	0.044	0.005	0.195	0.258	sphere ,	0.957317	0.01	0,	4.445
17	2.715	0.262	0.323	0.042	0.005	0.23	0.249	sphere ,	0.975460	0.009	0,	3.847
18	5.388	0.279	0.339	0.043	0.007	0.162	0.268	sphere ,	0.921687	0.013	0,	6.499
19	2.401	0.28	0.32	0.046	0.004	0.156	0.263	sphere ,	0.975155	0.01	0,	3.481
20	0.591	0.261	0.404	0.037	0.006	0.329	0.291	sphere ,	0.961783	0.013	0,	1.933
21	0.624	0.256	0.367	0.045	0.004	0.226	0.32	sphere ,	0.918239	0.052	0,	1.896
22	1.162	0.285	0.483	0.045	0.005	0.209	0.275	sphere ,	0.987578	0.01	0,	2.477
23	0.62	0.744	0.39	0.035	0.008	0.196	0.262	sphere ,	0.926380	0.01	0,	2.285
24	0.702	0.257	0.351	0.046	0.009	0.19	0.272	sphere ,	0.951220	0.015	0,	1.842
25	0.988	0.319	0.357	0.045	0.008	0.258	0.482	sphere ,	0.939759	0.018	0,	2.477
26	0.803	0.293	0.351	0.031	0.006	0.174	0.247	sphere ,	0.975155	0.01	0,	1.915
27	0.591	0.261	0.333	0.051	0.009	0.185	0.283	sphere ,	0.968354	0.021	0,	1.734
28	0.609	0.269	0.352	0.033	0.019	0.169	0.275	sphere ,	0.963855	0.007	0,	1.734
29	0.967	0.306	0.331	0.041	0.003	0.198	0.271	sphere ,	0.993750	0.013	0,	2.132
30	1.522	0.268	0.315	0.043	0.004	0.173	0.278	sphere ,	0.962025	0.018	0,	2.621
31	0.684	0.445	0.388	0.043	0.005	0.176	0.263	sphere ,	1.000000	0.013	0,	2.018
32	0.659	0.286	0.384	0.046	0.005	0.238	0.264	sphere ,	0.987421	0.01	0,	1.893
33	0.808	0.343	0.365	0.049	0.006	0.207	0.347	sphere ,	0.974684	0.039	0,	2.164
34	0.797	0.339	0.367	0.048	0.006	0.209	0.282	sphere ,	0.962963	0.012	0,	2.062
35	1.21	0.273	0.379	0.035	0.005	0.197	0.317	sphere ,	0.981250	0.01	0,	2.427
36	0.93	0.377	0.406	0.046	0.007	0.214	0.267	sphere ,	0.981366	0.02	0,	2.269
37	0.697	0.28	0.386	0.049	0.006	0.192	0.265	sphere ,	0.968750	0.021	0,	1.896
38	0.679	0.251	0.382	0.043	0.005	0.182	0.257	sphere ,	0.948718	0.01	0,	1.809
39	0.666	0.315	0.379	0.043	0.004	0.21	0.27	sphere ,	0.981481	0.008	0,	1.898
40	0.784	0.313	0.346	0.044	0.006	0.177	0.271	sphere ,	0.987578	0.022	0,	1.963
41	0.688	0.268	0.363	0.036	0.007	0.187	0.261	sphere ,	0.969325	0.011	0,	1.823
42	0.609	0.264	0.378	0.033	0.013	0.14	0.263	sphere ,	0.927711	0.009	0,	1.711
43	0.595	0.259	0.353	0.042	0.004	0.196	0.344	sphere ,	0.981366	0.013	0,	1.807
44	0.84	0.318	0.331	0.044	0.007	0.199	0.25	sphere ,	0.987730	0.009	0,	1.998
45	0.627	0.261	0.384	0.04	0.011	0.169	0.274	sphere ,	0.833333	0.011	0,	1.778
46	0.658	0.267	0.377	0.048	0.005	0.21	0.251	sphere ,	0.962500	0.009	0,	1.827
47	2.768	0.28	0.369	0.051	0.005	0.197	0.264	sphere ,	0.981707	0.009	0,	3.944
48	1.827	0.27	0.321	0.041	0.004	0.218	0.263	sphere ,	0.975460	0.02	0,	2.966
49	5.073	0.268	0.32	0.036	0.006	0.199	0.249	sphere ,	0.987805	0.012	0,	6.164
50	3.211	0.256	0.336	0.032	0.006	0.202	0.272	sphere ,	0.975610	0.022	0,	4.338
51	2.548	0.252	0.325	0.032	0.006	0.199	0.271	sphere ,	0.993865	0.018	0,	3.652
52	4.842	0.259	0.335	0.043	0.005	0.202	0.263	sphere ,	0.957317	0.01	0,	5.959
53	0.712	0.481	0.389	0.045	0.007	0.194	0.252	sphere ,	0.974843	0.01	0,	2.091
54	0.688	0.312	0.64	0.107	0.036	0.294	0.322	sphere ,	0.975309	0.009	0,	2.41
55	0.707	0.431	0.49	0.048	0.005	0.165	0.246	sphere ,	0.963636	0.021	0,	2.113
56	0.698	0.273	0.395	0.07	0.006	0.528	0.325	sphere ,	0.963415	0.011	0,	2.307
57	0.641	0.275	0.381	0.031	0.006	0.181	0.269	sphere ,	0.981366	0.014	0,	1.798
58	0.754	0.533	0.369	0.032	0.005	0.249	0.274	sphere ,	0.993711	0.011	0,	2.227

59	0.763	0.293	0.338	0.032	0.01	0.185	0.273	sphere ,	0.981366	0.012	0,	1.906
60	0.69	0.275	0.343	0.042	0.005	0.197	0.223	sphere ,	0.962264	0.01	0,	1.786
61	0.719	0.283	0.373	0.043	0.017	0.191	0.352	sphere ,	0.987421	0.024	0,	2.002
62	0.927	0.29	0.327	0.044	0.006	0.212	0.285	sphere ,	0.975610	0.012	0,	2.103
63	0.698	0.269	0.359	0.043	0.004	0.221	0.263	sphere ,	0.993750	0.009	0,	1.867
64	0.915	0.281	0.307	0.039	0.003	0.203	0.242	sphere ,	0.987500	0.009	0,	1.999
65	3.132	0.255	0.322	0.043	0.006	0.19	0.242	sphere ,	0.962963	0.009	0,	4.2
66	2.004	0.247	0.327	0.044	0.005	0.159	0.272	sphere ,	0.968553	0.009	0,	3.07
67	14.891	0.253	0.337	0.036	0.006	0.177	0.233	sphere ,	0.893082	0.009	0,	15.942
68	2.379	0.273	0.338	0.032	0.007	0.216	0.272	sphere ,	0.912500	0.012	0,	3.529
69	2.387	0.254	0.329	0.029	0.004	0.184	0.267	sphere ,	0.987500	0.012	0,	3.467
70	0.595	0.257	0.369	0.041	0.015	0.192	0.272	sphere ,	0.962025	0.011	0,	1.753
71	1.437	0.264	0.327	0.033	0.005	0.187	0.247	sphere ,	0.993827	0.008	0,	2.509
72	5.113	0.257	0.326	0.032	0.004	0.221	0.259	sphere ,	0.975758	0.013	0,	6.226
73	2.616	0.265	0.334	0.043	0.006	0.198	0.237	sphere ,	0.955975	0.01	0,	3.71
74	0.659	0.258	0.361	0.041	0.005	0.188	0.258	sphere ,	0.884146	0.011	0,	1.782
75	0.691	0.266	0.366	0.045	0.005	0.179	0.278	sphere ,	0.969512	0.01	0,	1.841
76	1.819	0.266	0.327	0.044	0.004	0.198	0.264	sphere ,	0.975000	0.009	0,	2.931
77	0.631	0.663	0.577	0.051	0.008	0.234	0.289	sphere ,	0.975460	0.011	0,	2.465
78	0.816	0.363	0.343	0.048	0.007	0.282	0.316	sphere ,	0.994220	0.032	0,	2.209
79	0.808	0.314	0.329	0.068	0.007	0.205	0.283	sphere ,	0.937500	0.012	0,	2.026
80	0.966	0.34	0.336	0.033	0.007	0.23	0.25	sphere ,	0.993827	0.021	0,	2.184
81	0.605	0.267	0.37	0.041	0.018	0.214	0.275	sphere ,	0.931677	0.05	0,	1.841
82	1.067	0.323	0.326	0.041	0.003	0.207	0.26	sphere ,	0.974684	0.013	0,	2.241
83	0.612	0.278	0.401	0.047	0.006	0.258	0.298	sphere ,	0.964072	0.012	0,	1.913
84	0.666	0.308	0.421	0.043	0.005	0.155	0.292	sphere ,	0.962733	0.013	0,	1.904
85	1.256	0.267	0.319	0.044	0.006	0.177	0.248	sphere ,	0.981481	0.012	0,	2.329
86	0.612	0.262	0.41	0.05	0.005	0.163	0.256	sphere ,	0.974843	0.007	0,	1.766
87	0.65	0.27	0.365	0.043	0.003	0.175	0.252	sphere ,	0.975155	0.009	0,	1.768
88	0.629	0.266	0.357	0.042	0.004	0.155	0.248	sphere ,	0.974359	0.009	0,	1.711
89	0.613	0.261	0.368	0.034	0.005	0.202	0.259	sphere ,	0.969325	0.01	0,	1.752
90	0.664	0.267	0.383	0.042	0.005	0.189	0.252	sphere ,	0.981013	0.009	0,	1.811
91	0.606	0.256	0.392	0.044	0.004	0.203	0.246	sphere ,	0.981481	0.008	0,	1.759
92	0.615	0.252	0.371	0.042	0.005	0.177	0.255	sphere ,	0.975000	0.011	0,	1.728
93	0.682	0.281	0.374	0.032	0.005	0.192	0.261	sphere ,	0.987578	0.019	0,	1.847
94	0.64	0.268	0.342	0.044	0.004	0.169	0.251	sphere ,	0.981707	0.009	0,	1.727
95	0.584	0.259	0.372	0.036	0.004	0.206	0.269	sphere ,	0.993827	0.02	0,	1.75
96	0.85	0.33	0.355	0.031	0.007	0.181	0.252	sphere ,	0.981250	0.018	0,	2.025
97	0.747	0.329	0.438	0.045	0.005	0.206	0.263	sphere ,	0.897590	0.01	0,	2.045
98	0.681	0.247	0.374	0.04	0.004	0.184	0.238	sphere ,	0.980769	0.021	0,	1.791
99	0.774	0.297	0.365	0.036	0.008	0.214	0.287	sphere ,	0.975904	0.019	0,	2
100	0.592	0.247	0.526	0.047	0.009	0.192	0.28	sphere ,	0.987342	0.011	0,	1.904

Código A.4: Datos obtenidos para la prueba de la esfera usando AGC.

1	0.578	0.242	0.349	0.062	0.684	0.267	4.065	cylinder	0.368601	0.039	1,	6.286
2	0.781	0.241	0.317	0.033	0.006	0.129	0.051	sphere	0.917808	0.012	1,	1.571
3	0.631	0.243	0.304	0.038	0.003	0.174	0.04	sphere	0.909722	0.026	1,	1.46
4	0.674	0.233	0.328	0.028	0.013	0.189	0.046	sphere	0.737931	0.024	1,	1.535
5	0.565	0.244	0.334	0.029	0.017	0.173	0.053	sphere	0.849315	0.013	1,	1.428
6	0.61	0.227	0.324	0.035	0.007	0.157	0.041	sphere	0.883562	0.014	1,	1.415
7	0.65	0.234	0.324	0.038	0.008	0.19	0.048	sphere	0.790210	0.013	1,	1.506
8	0.62	0.245	0.289	0.038	0.006	0.179	0.048	sphere	0.853147	0.015	1,	1.441
9	0.576	0.23	0.339	0.035	0.018	0.203	0.044	sphere	0.897260	0.024	1,	1.47
10	0.684	0.258	0.35	0.04	0.008	0.222	0.047	sphere	0.801370	0.028	1,	1.637
11	10.185	0.292	0.398	0.043	0.008	0.193	0.084	sphere	0.836735	0.031	1,	11.234
12	2.224	0.314	0.472	0.044	0.011	0.236	0.062	sphere	0.816327	0.016	1,	3.38
13	2.427	0.289	0.373	0.034	0.007	0.236	0.051	sphere	0.903448	0.02	1,	3.437
14	1.181	0.296	0.367	0.044	0.006	0.243	0.048	sphere	0.892617	0.027	1,	2.212
15	0.666	0.263	0.343	0.032	0.008	0.179	0.045	sphere	0.930070	0.025	1,	1.562

16	0.637	0.278	0.348	0.062	0.006	0.192	0.059	sphere	0.885135	0.015	1,	1.598
17	0.711	0.287	0.357	0.045	0.006	0.195	0.051	sphere	0.909722	0.017	1,	1.669
18	0.846	0.28	0.35	0.04	0.01	0.214	0.047	sphere	0.772414	0.014	1,	1.801
19	0.726	0.269	0.344	0.042	0.007	0.184	0.049	sphere	0.818182	0.02	1,	1.644
20	0.682	0.27	0.355	0.035	0.008	0.221	0.046	sphere	0.891156	0.024	1,	1.643
21	3.099	0.273	0.379	0.043	0.005	0.195	0.049	sphere	0.889655	0.016	1,	4.059
22	0.713	0.348	0.393	0.047	0.01	0.168	0.047	sphere	0.797297	0.018	1,	1.745
23	0.641	0.267	0.371	0.045	0.009	0.221	0.05	sphere	0.790541	0.015	1,	1.619
24	0.65	0.324	0.389	0.042	0.005	0.206	0.048	sphere	0.887324	0.018	1,	1.683
25	0.686	0.27	0.496	0.102	0.071	0.275	0.08	sphere	0.862069	0.024	1,	2.007
26	1.076	0.277	0.756	0.076	0.01	0.319	0.043	sphere	0.897260	0.032	1,	2.589
27	2.141	0.312	0.36	0.051	0.026	0.205	0.053	sphere	0.864865	0.025	1,	3.173
28	0.678	0.300	0.586	0.134	0.032	0.256	0.084	sphere	0.857143	0.019	1,	2.089
29	2.12	0.308	0.400	0.036	0.007	0.260	0.054	sphere	0.815068	0.035	1,	3.221
30	1.015	0.317	0.405	0.046	0.02	0.227	0.055	sphere	0.820690	0.02	1,	2.105
31	3.271	0.439	0.585	0.075	0.026	0.268	0.206	sphere	0.909722	0.063	1,	4.935
32	2.397	0.365	0.43	0.044	0.025	0.447	0.058	sphere	0.898649	0.016	1,	3.782
33	1.091	0.327	0.444	0.051	0.01	0.195	0.056	sphere	0.800000	0.016	1,	2.19
34	0.926	0.357	0.392	0.047	0.009	0.215	0.074	sphere	0.818792	0.039	1,	2.06
35	1.38	0.311	0.448	0.048	0.004	0.273	0.053	sphere	0.918919	0.057	1,	2.576
36	1.262	0.319	0.42	0.058	0.012	0.278	0.053	sphere	0.780822	0.039	1,	2.442
37	1.461	0.365	0.394	0.035	0.016	0.226	0.055	sphere	0.866197	0.017	1,	2.57
38	0.801	0.292	0.368	0.033	0.007	0.214	0.053	sphere	0.885135	0.027	1,	1.796
39	4.491	0.275	0.336	0.044	0.007	0.177	0.051	sphere	0.827586	0.025	1,	5.406
40	5.539	0.258	0.316	0.043	0.007	0.211	0.057	sphere	0.867550	0.017	1,	6.448
41	0.751	0.297	0.334	0.106	0.03	0.251	0.062	sphere	0.885906	0.025	1,	1.856
42	0.653	0.343	0.52	0.047	0.008	0.199	0.039	sphere	0.868056	0.019	1,	1.83
43	0.606	0.281	0.345	0.032	0.008	0.205	0.071	sphere	0.790541	0.044	1,	1.594
44	1.846	0.296	0.39	0.043	0.005	0.316	0.053	sphere	0.913333	0.03	1,	2.979
45	0.659	0.295	0.358	0.045	0.008	0.236	0.057	sphere	0.853333	0.019	1,	1.679
46	1.415	0.268	0.335	0.044	0.004	0.208	0.053	sphere	0.919463	0.015	1,	2.342
47	0.82	0.287	0.550	0.054	0.009	0.201	0.058	sphere	0.835526	0.017	1,	1.996
48	0.822	0.272	0.365	0.042	0.007	0.152	0.047	sphere	0.883562	0.017	1,	1.724
49	0.719	0.278	0.365	0.045	0.007	0.205	0.038	sphere	0.858108	0.024	1,	1.681
50	0.776	0.295	0.550	0.058	0.007	0.237	0.044	sphere	0.846667	0.026	1,	1.993
51	1.23	0.56	0.759	0.037	0.007	0.242	0.106	sphere	0.926174	0.018	1,	2.979
52	0.924	0.323	0.346	0.045	0.007	0.253	0.047	sphere	0.826667	0.023	1,	1.968
53	0.746	0.287	0.427	0.087	0.014	0.207	0.055	sphere	0.885906	0.02	1,	1.843
54	0.649	0.255	0.379	0.045	0.007	0.239	0.039	sphere	0.866667	0.029	1,	1.642
55	0.625	0.262	0.396	0.034	0.016	0.203	0.047	sphere	0.868966	0.018	1,	1.601
56	1.098	0.288	0.332	0.034	0.008	0.222	0.079	sphere	0.865772	0.014	1,	2.075
57	0.63	0.263	0.418	0.033	0.006	0.179	0.049	sphere	0.882759	0.018	1,	1.598
58	0.676	0.284	0.403	0.035	0.016	0.177	0.04	sphere	0.800000	0.024	1,	1.655
59	0.771	0.291	0.350	0.03	0.016	0.154	0.046	sphere	0.897260	0.018	1,	1.677
60	0.624	0.253	0.363	0.047	0.01	0.183	0.052	sphere	0.907285	0.016	1,	1.548
61	0.664	0.272	0.402	0.042	0.009	0.18	0.048	sphere	0.843537	0.014	1,	1.633
62	0.714	0.326	0.370	0.028	0.007	0.183	0.034	sphere	0.855172	0.018	1,	1.682
63	0.746	0.258	0.368	0.029	0.008	0.189	0.054	sphere	0.821192	0.013	1,	1.665
64	0.599	0.262	0.382	0.044	0.006	0.225	0.056	sphere	0.852349	0.017	1,	1.592
65	0.685	0.282	0.367	0.041	0.007	0.247	0.053	sphere	0.815789	0.017	1,	1.701
66	0.607	0.288	0.386	0.044	0.006	0.17	0.056	sphere	0.844595	0.013	1,	1.572
67	0.602	0.252	0.379	0.041	0.006	0.212	0.057	sphere	0.901961	0.027	1,	1.585
68	0.682	0.276	0.374	0.045	0.005	0.317	0.059	sphere	0.893333	0.018	1,	1.776
69	0.684	0.269	0.370	0.036	0.014	0.18	0.056	sphere	0.873333	0.016	1,	1.625
70	0.673	0.318	0.400	0.034	0.009	0.218	0.047	sphere	0.765101	0.014	1,	1.713
71	1.356	0.272	0.375	0.043	0.008	0.165	0.051	sphere	0.833333	0.032	1,	2.304
72	0.654	0.469	1.022	0.204	0.034	0.258	0.063	sphere	0.781457	0.016	1,	2.721
73	0.754	0.482	0.375	0.033	0.004	0.235	0.06	sphere	0.907285	0.015	1,	1.958
74	0.654	0.287	0.387	0.043	0.012	0.234	0.065	sphere	0.856209	0.021	1,	1.704
75	0.691	0.281	0.409	0.044	0.009	0.214	0.068	sphere	0.784314	0.026	1,	1.742
76	1.138	0.286	0.351	0.043	0.008	0.152	0.049	sphere	0.751678	0.031	1,	2.058
77	0.842	0.538	0.503	0.06	0.007	0.298	0.052	sphere	0.904110	0.038	1,	2.339
78	0.98	0.304	0.381	0.046	0.009	0.265	0.148	sphere	0.853333	0.04	1,	2.173
79	0.757	0.347	0.446	0.056	0.019	0.221	0.053	sphere	0.860927	0.018	1,	1.918

80	0.699	0.296	0.396	0.036	0.006	0.27	0.058	sphere	0.895425	0.024	1,	1.793
81	0.843	0.341	0.371	0.051	0.007	0.254	0.06	sphere	0.870130	0.027	1,	1.955
82	0.737	0.289	0.362	0.043	0.008	0.192	0.055	sphere	0.812500	0.015	1,	1.701
83	0.716	0.262	0.386	0.044	0.018	0.155	0.048	sphere	0.813793	0.016	1,	1.645
84	2.355	0.259	0.353	0.033	0.01	0.237	0.041	sphere	0.789474	0.028	1,	3.317
85	0.773	0.311	0.384	0.045	0.007	0.226	0.068	sphere	0.842105	0.017	1,	1.831
86	0.662	0.26	0.394	0.042	0.007	0.228	0.059	sphere	0.902597	0.016	1,	1.668
87	0.617	0.258	0.36	0.032	0.006	0.203	0.046	sphere	0.860927	0.027	1,	1.558
88	0.704	0.261	0.36	0.037	0.009	0.221	0.051	sphere	0.900662	0.015	1,	1.659
89	0.653	0.267	0.382	0.047	0.004	0.235	0.054	sphere	0.947020	0.017	1,	1.66
90	0.598	0.265	0.405	0.046	0.005	0.226	0.048	sphere	0.928571	0.016	1,	1.61
91	0.681	0.276	0.376	0.041	0.005	0.19	0.051	sphere	0.909722	0.016	1,	1.637
92	0.656	0.25	0.382	0.042	0.007	0.235	0.048	sphere	0.860927	0.015	1,	1.635
93	0.604	0.264	0.362	0.045	0.017	0.202	0.059	sphere	0.803922	0.015	1,	1.569
94	0.618	0.276	0.369	0.031	0.005	0.232	0.047	sphere	0.896104	0.017	1,	1.596
95	0.725	0.267	0.363	0.038	0.006	0.221	0.051	sphere	0.890411	0.019	1,	1.69
96	0.628	0.268	0.408	0.042	0.006	0.188	0.056	sphere	0.860000	0.017	1,	1.613
97	0.654	0.272	0.405	0.04	0.017	0.221	0.053	sphere	0.870968	0.015	1,	1.678
98	0.974	0.291	0.337	0.045	0.009	0.226	0.052	sphere	0.738255	0.014	1,	1.948
99	0.619	0.269	0.379	0.042	0.006	0.224	0.052	sphere	0.904762	0.015	1,	1.606
100	0.742	0.288	0.355	0.03	0.006	0.205	0.05	sphere	0.922078	0.017	1,	1.694

Código A.5: Datos obtenidos para la prueba del cilindro usando AE.

1	0.576	0.248	0.32	0.016	0.01	0.006	0.051	cylinder ,	0.807692	0.011	0,	1.238
2	0.664	0.243	0.331	0.015	0.015	0.014	0.062	sphere ,	0.584906	0.008	0,	1.352
3	0.59	0.228	0.316	0.017	0.017	0.006	0.055	cylinder ,	0.647059	0.01	0,	1.239
4	0.58	0.246	0.298	0.014	0.017	0.009	0.056	cylinder ,	0.653846	0.012	0,	1.233
5	0.665	0.243	0.317	0.015	0.021	0.007	0.056	sphere ,	0.603774	0.01	0,	1.336
6	0.901	0.239	0.315	0.016	0.016	0.005	0.056	cylinder ,	0.764706	0.009	0,	1.559
7	0.659	0.276	0.412	0.015	0.012	0.016	0.11	sphere ,	0.600000	0.01	0,	1.511
8	0.76	0.297	0.33	0.03	0.018	0.01	0.065	cylinder ,	0.592593	0.011	0,	1.521
9	0.755	0.258	0.362	0.032	0.036	0.025	0.088	sphere ,	0.559322	0.014	0,	1.571
10	0.848	0.487	0.359	0.027	0.026	0.033	0.151	cylinder ,	0.548387	0.036	0,	1.968
11	0.867	0.314	0.354	0.029	0.014	0.01	0.182	sphere ,	0.576271	0.01	0,	1.781
12	0.664	0.631	0.368	0.03	0.013	0.011	0.072	cylinder ,	0.634615	0.018	0,	1.808
13	0.612	0.278	0.505	0.061	0.032	0.016	0.08	cylinder ,	0.629630	0.025	0,	1.61
14	0.705	0.28	0.361	0.021	0.028	0.008	0.267	plane ,	0.580000	0.009	0,	1.679
15	0.731	0.279	0.357	0.021	0.028	0.013	0.077	cylinder ,	0.622951	0.019	0,	1.525
16	0.662	0.277	0.363	0.029	0.025	0.013	0.092	plane ,	0.540984	0.008	0,	1.469
17	0.711	0.328	0.337	0.015	0.016	0.009	0.059	cylinder ,	0.666667	0.023	0,	1.498
18	0.631	0.361	0.511	0.054	0.029	0.08	0.098	sphere ,	0.576271	0.013	0,	1.778
19	0.817	0.279	0.376	0.031	0.012	0.007	0.057	cylinder ,	0.884615	0.012	0,	1.591
20	0.622	0.258	0.352	0.029	0.016	0.008	0.06	cylinder ,	0.622642	0.014	0,	1.36
21	0.695	0.44	0.338	0.025	0.012	0.006	0.053	cylinder ,	0.705882	0.012	0,	1.582
22	0.641	0.257	0.342	0.025	0.027	0.011	0.07	cylinder ,	0.596154	0.02	0,	1.396
23	0.919	0.281	0.365	0.012	0.021	0.024	0.075	sphere ,	0.566667	0.008	0,	1.707
24	0.627	0.454	0.382	0.016	0.015	0.017	0.053	plane ,	0.603774	0.021	0,	1.589
25	0.597	0.278	0.336	0.027	0.02	0.009	0.064	cylinder ,	0.611111	0.011	0,	1.344
26	1.416	0.267	0.364	0.016	0.019	0.009	0.072	sphere ,	0.576271	0.01	0,	2.174
27	0.76	0.288	0.347	0.016	0.033	0.008	0.058	plane ,	0.588235	0.011	0,	1.521
28	1.162	0.274	0.349	0.018	0.026	0.01	0.081	sphere ,	0.532258	0.011	0,	1.931
29	0.846	0.343	0.386	0.04	0.018	0.02	0.073	cylinder ,	0.792453	0.01	0,	1.739
30	0.644	0.315	0.513	0.02	0.053	0.005	0.128	plane ,	0.620000	0.011	0,	1.689
31	0.64	0.28	0.337	0.031	0.017	0.007	0.063	cylinder ,	0.647059	0.012	0,	1.389
32	0.674	0.266	0.358	0.017	0.015	0.007	0.06	sphere ,	0.576923	0.009	0,	1.407
33	8.211	0.267	0.343	0.022	0.02	0.007	0.057	sphere ,	0.603774	0.01	0,	8.938
34	2.16	0.268	0.33	0.029	0.012	0.007	0.068	cylinder ,	0.581818	0.01	0,	2.887
35	3.28	0.255	0.341	0.014	0.019	0.008	0.08	sphere ,	0.603774	0.013	0,	4.01
36	2.95	0.25	0.337	0.029	0.018	0.007	0.062	cylinder ,	0.647059	0.013	0,	3.666

37	0.632	0.262	0.354	0.019	0.016	0.007	0.066	cylinder ,	0.660377	0.011	0,	1.369
38	1.58	0.268	0.384	0.023	0.021	0.01	0.081	sphere ,	0.559322	0.02	0,	2.389
39	0.947	0.297	0.38	0.016	0.02	0.011	0.059	cylinder ,	0.596154	0.013	0,	1.743
40	0.726	0.471	0.482	0.018	0.016	0.007	0.069	cylinder ,	0.584906	0.011	0,	1.8
41	0.715	0.248	0.407	0.015	0.027	0.01	0.074	cylinder ,	0.563636	0.01	0,	1.508
42	0.781	0.395	0.374	0.02	0.016	0.007	0.082	plane ,	0.580000	0.01	0,	1.685
43	0.605	0.326	0.726	0.017	0.023	0.007	0.071	cylinder ,	0.851852	0.013	0,	1.789
44	0.613	0.267	0.347	0.021	0.016	0.008	0.057	cylinder ,	0.666667	0.012	0,	1.344
45	0.721	0.242	0.355	0.019	0.025	0.008	0.056	cylinder ,	0.666667	0.013	0,	1.44
46	0.682	0.268	0.343	0.022	0.027	0.007	0.063	cylinder ,	0.574074	0.013	0,	1.425
47	0.603	0.263	0.33	0.028	0.021	0.017	0.066	cylinder ,	0.555556	0.015	0,	1.344
48	1.159	0.271	0.354	0.028	0.017	0.013	0.074	plane ,	0.566038	0.006	0,	1.924
49	0.637	0.266	0.451	0.017	0.022	0.011	0.066	cylinder ,	0.566038	0.026	0,	1.496
50	0.652	0.276	0.368	0.02	0.018	0.018	0.069	sphere ,	0.555556	0.01	0,	1.433
51	0.681	0.272	0.473	0.022	0.018	0.013	0.405	sphere ,	0.588235	0.034	0,	1.918
52	0.833	0.287	0.354	0.019	0.017	0.021	0.066	sphere ,	0.566038	0.008	0,	1.606
53	0.596	0.282	0.358	0.033	0.016	0.01	0.062	cylinder ,	0.685185	0.011	0,	1.368
54	0.621	0.248	0.322	0.032	0.023	0.011	0.099	sphere ,	0.540984	0.011	0,	1.369
55	0.685	0.274	0.364	0.019	0.022	0.009	0.066	cylinder ,	0.622642	0.019	0,	1.459
56	0.674	0.254	0.347	0.032	0.024	0.01	0.101	plane ,	0.539683	0.017	0,	1.46
57	0.561	0.27	0.359	0.019	0.026	0.009	0.056	cylinder ,	0.611111	0.013	0,	1.313
58	0.561	0.261	0.353	0.016	0.015	0.01	0.068	cylinder ,	0.673077	0.02	0,	1.307
59	0.584	0.261	0.36	0.029	0.028	0.009	0.079	plane ,	0.563636	0.015	0,	1.365
60	0.605	0.271	0.352	0.023	0.016	0.011	0.064	sphere ,	0.592593	0.012	0,	1.354
61	0.708	0.261	0.335	0.04	0.023	0.012	0.065	cylinder ,	0.592593	0.022	0,	1.468
62	0.594	0.27	0.359	0.026	0.03	0.008	0.055	cylinder ,	0.666667	0.011	0,	1.355
63	0.62	0.266	0.35	0.018	0.018	0.016	0.05	cylinder ,	0.788462	0.021	0,	1.361
64	0.629	0.266	0.352	0.02	0.028	0.009	0.064	cylinder ,	0.796296	0.012	0,	1.38
65	0.709	0.273	0.323	0.014	0.023	0.007	0.073	plane ,	0.557692	0.007	0,	1.429
66	0.629	0.252	0.368	0.016	0.024	0.01	0.084	plane ,	0.542373	0.008	0,	1.391
67	0.626	0.26	0.353	0.03	0.017	0.008	0.083	cylinder ,	0.612903	0.021	0,	1.399
68	0.776	0.289	0.36	0.037	0.018	0.01	0.079	cylinder ,	0.547170	0.015	0,	1.584
69	0.647	0.264	0.353	0.018	0.027	0.007	0.056	cylinder ,	0.547170	0.015	0,	1.387
70	0.598	0.268	0.341	0.02	0.025	0.008	0.063	plane ,	0.537037	0.008	0,	1.331
71	0.647	0.266	0.345	0.022	0.026	0.015	0.067	sphere ,	0.574074	0.01	0,	1.398
72	1.089	0.331	0.351	0.027	0.022	0.013	0.071	cylinder ,	0.615385	0.018	0,	1.922
73	0.997	0.497	0.795	0.044	0.033	0.013	0.145	sphere ,	0.539683	0.056	0,	2.581
74	1.111	0.293	0.351	0.029	0.012	0.012	0.067	cylinder ,	0.629630	0.016	0,	1.892
75	0.695	0.256	0.422	0.018	0.026	0.015	0.129	sphere ,	0.532258	0.058	0,	1.62
76	0.711	0.268	0.421	0.028	0.02	0.007	0.068	plane ,	0.584906	0.008	0,	1.532
77	0.983	0.400	0.357	0.02	0.02	0.008	0.086	plane ,	0.589286	0.007	0,	1.882
78	0.716	0.25	0.375	0.019	0.022	0.013	0.082	sphere ,	0.622642	0.012	0,	1.489
79	0.731	0.261	0.538	0.019	0.026	0.019	0.08	sphere ,	0.532258	0.011	0,	1.688
80	0.784	0.263	0.539	0.036	0.282	0.019	0.114	sphere ,	0.573770	0.01	0,	2.047
81	0.631	0.275	0.400	0.021	0.068	0.017	0.073	plane ,	0.576923	0.008	0,	1.494
82	0.906	0.303	0.393	0.016	0.019	0.028	0.068	cylinder ,	0.666667	0.011	0,	1.745
83	0.705	0.296	0.411	0.015	0.015	0.006	0.067	plane ,	0.611111	0.009	0,	1.526
84	1.111	0.285	0.346	0.029	0.022	0.011	0.088	sphere ,	0.516667	0.008	0,	1.902
85	0.659	0.437	0.607	0.034	0.027	0.029	0.079	cylinder ,	0.524590	0.014	0,	1.887
86	0.573	0.27	0.36	0.018	0.02	0.008	0.078	sphere ,	0.559322	0.008	0,	1.336
87	0.622	0.306	0.372	0.019	0.022	0.015	0.067	cylinder ,	0.573770	0.012	0,	1.437
88	0.873	0.359	0.388	0.031	0.036	0.01	0.066	plane ,	0.547170	0.007	0,	1.771
89	0.888	0.314	0.384	0.028	0.024	0.01	0.102	sphere ,	0.533333	0.012	0,	1.763
90	0.625	0.273	0.36	0.029	0.014	0.009	0.071	sphere ,	0.557692	0.014	0,	1.395
91	1.149	0.322	0.411	0.036	0.027	0.022	0.100	cylinder ,	0.508197	0.025	0,	2.093
92	0.614	0.274	0.536	0.089	0.041	0.128	0.205	plane ,	0.600000	0.065	0,	1.954
93	0.743	0.272	0.353	0.027	0.023	0.01	0.164	plane ,	0.540000	0.007	0,	1.601
94	0.688	0.291	0.411	0.022	0.02	0.012	0.069	sphere ,	0.559322	0.013	0,	1.526
95	0.758	0.277	0.424	0.015	0.029	0.009	0.059	cylinder ,	0.557692	0.018	0,	1.591
96	0.997	0.319	0.337	0.027	0.014	0.009	0.071	sphere ,	0.600000	0.01	0,	1.784
97	0.677	0.275	0.379	0.021	0.026	0.02	0.08	plane ,	0.516667	0.008	0,	1.486
98	0.655	0.266	0.354	0.03	0.028	0.037	0.118	sphere ,	0.526316	0.03	0,	1.518
99	0.723	0.375	0.333	0.018	0.019	0.011	0.087	sphere ,	0.573770	0.009	0,	1.576
100	0.771	0.447	0.358	0.018	0.017	0.018	0.086	sphere ,	0.553571	0.01	0,	1.727

Código A.6: Datos obtenidos para la prueba del cilindro usando AGC.

1	0.824	0.282	0.349	0.016	0.055	0.009	0.178	cylinder	0.788462	0.031	1,	1.745
2	0.728	0.264	0.365	0.019	0.047	0.022	0.221	cylinder	0.654545	0.041	1,	1.707
3	0.81	0.281	0.346	0.015	0.036	0.009	0.244	cylinder	0.800000	0.041	1,	1.783
4	0.755	0.284	0.322	0.014	0.048	0.01	0.200	cylinder	0.703704	0.038	1,	1.672
5	0.739	0.288	0.37	0.015	0.023	0.017	0.196	cylinder	0.923077	0.04	1,	1.688
6	0.803	0.272	0.352	0.029	0.032	0.017	0.239	cylinder	0.730769	0.03	1,	1.776
7	0.783	0.323	0.355	0.017	0.051	0.01	0.185	cylinder	0.545455	0.029	1,	1.754
8	0.891	0.300	0.34	0.019	0.028	0.009	0.13	cylinder	0.555556	0.04	1,	1.758
9	0.794	0.279	0.341	0.018	0.04	0.01	0.223	cylinder	0.696429	0.029	1,	1.734
10	0.832	0.308	0.365	0.017	0.045	0.009	0.312	cylinder	0.833333	0.036	1,	1.925
11	0.766	0.285	0.42	0.032	0.071	0.009	0.390	plane	0.527273	0.055	1,	2.029
12	0.897	0.317	0.574	0.046	0.049	0.009	0.301	cylinder	0.578947	0.054	1,	2.248
13	0.988	0.315	0.341	0.027	0.039	0.023	0.224	cylinder	0.636364	0.04	1,	1.999
14	0.846	0.346	0.732	0.026	0.054	0.016	0.278	cylinder	0.870370	0.027	1,	2.327
15	0.745	0.329	0.454	0.02	0.06	0.01	0.234	cylinder	0.727273	0.029	1,	1.881
16	1.224	0.276	0.352	0.023	0.042	0.012	0.269	cylinder	0.910714	0.044	1,	2.243
17	0.713	0.527	0.355	0.028	0.029	0.025	0.242	cylinder	0.490566	0.043	1,	1.962
18	0.739	0.279	0.336	0.019	0.061	0.012	0.20	cylinder	0.566038	0.042	1,	1.688
19	0.867	0.282	0.366	0.016	0.046	0.007	0.212	cylinder	0.886792	0.03	1,	1.826
20	0.798	0.269	0.333	0.016	0.042	0.02	0.22	cylinder	0.769231	0.032	1,	1.73
21	4.802	0.243	0.357	0.021	0.061	0.008	0.176	cylinder	0.777778	0.029	1,	5.698
22	1.959	0.265	0.355	0.016	0.053	0.011	0.209	cylinder	0.788462	0.031	1,	2.899
23	2.556	0.27	0.355	0.018	0.038	0.011	0.188	cylinder	0.535714	0.032	1,	3.47
24	1.659	0.256	0.371	0.021	0.039	0.013	0.23	cylinder	0.654545	0.04	1,	2.629
25	3.002	0.265	0.334	0.019	0.04	0.009	0.215	cylinder	0.517857	0.037	1,	3.923
26	3.871	0.26	0.317	0.021	0.029	0.02	0.192	cylinder	0.537037	0.028	1,	4.74
27	0.757	0.278	0.376	0.028	0.024	0.009	0.212	cylinder	0.576923	0.041	1,	1.725
28	0.751	0.277	0.366	0.017	0.057	0.02	0.231	cylinder	0.884615	0.029	1,	1.748
29	0.805	0.356	0.363	0.018	0.022	0.011	0.236	cylinder	0.576923	0.033	1,	1.844
30	1.165	0.278	0.336	0.018	0.037	0.014	0.214	cylinder	0.500000	0.041	1,	2.104
31	0.83	0.300	0.318	0.02	0.045	0.011	0.183	cylinder	0.678571	0.03	1,	1.739
32	0.705	0.269	0.365	0.029	0.023	0.011	0.229	cylinder	0.679245	0.043	1,	1.675
33	0.735	0.291	0.464	0.016	0.037	0.008	0.225	cylinder	0.921569	0.034	1,	1.81
34	0.776	0.284	0.36	0.019	0.042	0.008	0.211	cylinder	0.944444	0.03	1,	1.73
35	0.702	0.306	0.495	0.036	0.076	0.04	0.303	cylinder	0.703704	0.144	1,	2.103
36	1.226	0.436	0.371	0.02	0.05	0.009	0.236	plane	0.527273	0.013	1,	2.361
37	0.855	0.286	0.352	0.025	0.053	0.019	0.248	cylinder	0.759259	0.028	1,	1.867
38	1.356	0.314	0.354	0.02	0.045	0.008	0.212	cylinder	0.924528	0.047	1,	2.356
39	0.78	0.286	0.346	0.018	0.043	0.011	0.24	cylinder	0.509091	0.04	1,	1.765
40	0.703	0.261	0.338	0.02	0.043	0.008	0.175	cylinder	0.679245	0.04	1,	1.588
41	0.728	0.267	0.36	0.027	0.308	0.024	0.263	cylinder	0.923077	0.043	1,	2.02
42	1.027	0.271	0.353	0.018	0.04	0.009	0.209	cylinder	0.924528	0.032	1,	1.961
43	0.785	0.275	0.322	0.019	0.051	0.009	0.232	cylinder	0.574074	0.034	1,	1.728
44	0.967	0.277	0.452	0.032	0.05	0.012	0.266	cylinder	0.519231	0.044	1,	2.1
45	0.746	0.276	0.373	0.044	0.045	0.011	0.417	cylinder	0.898305	0.20	1,	2.112
46	1.207	0.287	0.402	0.017	0.038	0.01	0.235	cylinder	0.818182	0.029	1,	2.226
47	1.054	0.307	0.357	0.031	0.031	0.013	0.214	cylinder	0.927273	0.041	1,	2.049
48	0.73	0.286	0.559	0.034	0.07	0.041	0.367	cylinder	0.543860	0.071	1,	2.159
49	0.915	0.364	0.342	0.018	0.049	0.01	0.255	cylinder	0.553571	0.033	1,	1.987
50	0.932	0.318	0.352	0.018	0.044	0.01	0.237	cylinder	0.872727	0.038	1,	1.949
51	0.75	0.272	0.382	0.018	0.047	0.038	0.255	cylinder	0.629630	0.043	1,	1.805
52	0.981	0.294	0.459	0.062	0.044	0.023	0.169	cylinder	0.981132	0.029	1,	2.061
53	0.739	0.499	0.382	0.018	0.047	0.023	0.27	cylinder	0.730769	0.033	1,	2.012
54	1.158	0.276	0.513	0.023	0.04	0.01	0.231	cylinder	0.584906	0.042	1,	2.294
55	0.737	0.268	0.388	0.02	0.048	0.041	0.566	cylinder	0.750000	0.129	1,	2.197
56	1.014	0.321	0.328	0.027	0.029	0.008	0.228	cylinder	0.884615	0.03	1,	1.986
57	0.715	0.346	0.396	0.016	0.039	0.01	0.201	cylinder	0.759259	0.034	1,	1.758
58	0.708	0.268	0.37	0.021	0.057	0.012	0.216	cylinder	0.781818	0.043	1,	1.696

59	1.285	0.294	0.344	0.018	0.04	0.01	0.227	cylinder	0.875000	0.046	1,	2.264
60	0.729	0.284	0.377	0.027	0.038	0.009	0.188	cylinder	0.913793	0.033	1,	1.685
61	0.836	0.369	0.363	0.019	0.026	0.02	0.169	cylinder	0.962963	0.032	1,	1.835
62	0.979	0.303	0.349	0.017	0.033	0.023	0.426	cylinder	0.629630	0.079	1,	2.209
63	0.794	0.293	0.346	0.036	0.056	0.01	0.169	cylinder	0.732143	0.031	1,	1.736
64	0.804	0.279	0.359	0.019	0.038	0.01	0.237	cylinder	0.607143	0.031	1,	1.777
65	0.700	0.292	0.369	0.016	0.043	0.009	0.225	cylinder	0.867925	0.04	1,	1.694
66	0.729	0.261	0.357	0.031	0.035	0.012	0.222	cylinder	0.571429	0.033	1,	1.68
67	0.795	0.287	0.382	0.016	0.042	0.007	0.229	plane	0.592593	0.013	1,	1.771
68	0.655	0.263	0.36	0.018	0.042	0.018	0.203	cylinder	0.596154	0.04	1,	1.599
69	0.733	0.276	0.364	0.018	0.051	0.01	0.194	cylinder	0.527273	0.041	1,	1.688
70	0.805	0.283	0.35	0.026	0.019	0.011	0.257	cylinder	0.545455	0.028	1,	1.781
71	0.698	0.295	0.363	0.029	0.024	0.008	0.197	cylinder	0.636364	0.033	1,	1.647
72	0.726	0.251	0.363	0.027	0.038	0.009	0.199	cylinder	0.696429	0.04	1,	1.654
73	0.809	0.279	0.337	0.029	0.026	0.025	0.171	cylinder	0.678571	0.044	1,	1.721
74	0.700	0.269	0.366	0.013	0.042	0.01	0.231	cylinder	0.910714	0.042	1,	1.673
75	0.736	0.277	0.366	0.017	0.024	0.01	0.238	cylinder	0.660714	0.039	1,	1.707
76	0.794	0.263	0.356	0.025	0.03	0.014	0.192	cylinder	0.763636	0.039	1,	1.713
77	0.729	0.27	0.357	0.018	0.045	0.012	0.192	cylinder	0.537037	0.032	1,	1.655
78	0.689	0.256	0.375	0.016	0.042	0.007	0.277	cylinder	0.528302	0.037	1,	1.7
79	0.811	0.286	0.359	0.016	0.039	0.009	0.239	cylinder	0.759259	0.027	1,	1.788
80	0.734	0.256	0.365	0.025	0.053	0.013	0.22	cylinder	0.777778	0.035	1,	1.701
81	0.765	0.271	0.389	0.028	0.023	0.019	0.199	cylinder	0.686275	0.046	1,	1.741
82	0.83	0.29	0.341	0.026	0.032	0.011	0.239	cylinder	0.636364	0.032	1,	1.801
83	0.726	0.253	0.379	0.015	0.049	0.013	0.161	cylinder	0.537037	0.04	1,	1.637
84	0.728	0.271	0.363	0.027	0.032	0.008	0.227	plane	0.566038	0.023	1,	1.679
85	0.67	0.281	0.37	0.019	0.039	0.01	0.246	cylinder	0.517857	0.029	1,	1.664
86	0.717	0.266	0.393	0.028	0.041	0.01	0.228	cylinder	0.634615	0.044	1,	1.727
87	0.884	0.462	0.395	0.019	0.05	0.008	0.224	cylinder	0.943396	0.045	1,	2.088
88	1.791	0.293	0.446	0.02	0.053	0.012	0.26	cylinder	0.745455	0.043	1,	2.918
89	0.883	0.38	0.394	0.027	0.036	0.019	0.235	cylinder	0.905660	0.027	1,	2.002
90	0.84	0.307	0.362	0.026	0.06	0.013	0.237	cylinder	0.803571	0.045	1,	1.891
91	1.358	0.284	0.427	0.048	0.066	0.009	0.237	cylinder	0.509434	0.039	1,	2.468
92	0.796	0.343	0.342	0.03	0.026	0.014	0.225	cylinder	0.526316	0.031	1,	1.808
93	1.228	0.305	0.456	0.028	0.082	0.01	0.286	cylinder	0.759259	0.032	1,	2.43
94	0.957	0.347	0.365	0.029	0.021	0.007	0.181	cylinder	0.849057	0.037	1,	1.945
95	0.708	0.257	0.395	0.016	0.042	0.009	0.172	cylinder	0.788462	0.035	1,	1.635
96	0.714	0.261	0.376	0.028	0.022	0.01	0.216	cylinder	0.836364	0.028	1,	1.656
97	0.804	0.265	0.368	0.03	0.036	0.017	0.200	cylinder	0.672727	0.04	1,	1.76
98	0.673	0.278	0.381	0.03	0.038	0.028	0.213	cylinder	0.654545	0.03	1,	1.671
99	0.69	0.255	0.37	0.017	0.035	0.009	0.195	cylinder	0.452830	0.044	1,	1.617
100	0.838	0.257	0.359	0.023	0.032	0.011	0.244	cylinder	0.800000	0.041	1,	1.806

B CÓDIGOS

En esta sección se muestran los códigos de los archivos desarrollados y modificados para desarrollar el proyecto. Hay que tener en cuenta que para usar estos códigos es necesario tener instaladas las bibliotecas y programas siguientes:

- PCL 1.8.1 [4]
- libfreenect2 [24]
- libfreenect2pclgrabber [25]
- Vensor [30]

Código B.7: Archivo test.cpp

```

1  /*
2  Copyright 2018, Carlos Villasenor
3  This program is free software: you can redistribute it and/or modify
4  it under the terms of the GNU General Public License as published by
5  the Free Software Foundation, either version 3 of the License, or
6  (at your option) any later version.
7  This program is distributed in the hope that it will be useful,
8  but WITHOUT ANY WARRANTY; without even the implied warranty of
9  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
10 GNU General Public License for more details.
11 You should have received a copy of the GNU General Public License
12 along with this program. If not, see <http://www.gnu.org/licenses/>.
13 @Author
14 Carlos Villasenor, Student
15 CIDETEC -IPN, CD. MX. Mexico
16 */
17 #include "k2g.h"
18 #include <pcl/visualization/cloud_viewer.h>
19 #include <chrono>
20 #include <pcl/segmentation/segment_differences.h>
21 #include <pcl/filters/voxel_grid.h>
22 #include <pcl/console/print.h>
23 #include <pcl/console/parse.h>
24 #include <pcl/console/time.h>
25 #include <pcl/io/ply_io.h>
26 #include <pcl/kdtree/kdtree.h>
27 #include <pcl/segmentation/extract_clusters.h>
28 #include <pcl/sample_consensus/ransac.h>
29 #include <pcl/sample_consensus/sac_model_plane.h>
30 #include <pcl/sample_consensus/sac_model_sphere.h>
31 #include <pcl/sample_consensus/sac_model_cylinder.h>
32 #include <pcl/features/normal_3d.h>
33 #include <pcl/filters/extract_indices.h>
34 #include <type_traits>
35 #include "detail/vsr_multivector.h"
36 #include "ransac.h"
37 #include <iostream>
38 #include <fstream>
39 #include <time.h>
40
41 using namespace std;
42 typedef pcl::PointXYZRGB PointType;
43 typedef pcl::Normal PointNType;
44
45 double tress = 0.001;
46 double down = 0.025;
47 bool _downSample = true;
48 bool _resta = true;
49 bool _cluster = false;
50 bool _ciclo = true;
51 bool _ciclo2 = false;
52 bool _useAgc = true;
53 bool _debug = true;
54
55 boost::shared_ptr<pcl::PointCloud<PointType>> cloudKinect;
56 pcl::PointCloud<PointType>::Ptr cloudAux(new pcl::PointCloud<PointType>);
57 pcl::PointCloud<PointType>::Ptr cloudAux2(new pcl::PointCloud<PointType>);
58 pcl::PointCloud<PointType>::Ptr cloudCopy(new pcl::PointCloud<PointType>);
59 pcl::PointCloud<PointType>::Ptr cloudOut(new pcl::PointCloud<PointType>);
60 // Creating the KdTree object for the search method of the extraction
61 pcl::search::KdTree<PointType>::Ptr tree(new pcl::search::KdTree<PointType>);

```

```

62
63
64 struct PlySaver {
65
66
67 PlySaver(boost::shared_ptr<pcl::PointCloud<PointType>> cloud, bool binary, bool use_camera, K2G & k2g) :
68 cloud_(cloud), binary_(binary), use_camera_(use_camera), k2g_(k2g) {}
69
70 boost::shared_ptr<pcl::PointCloud<PointType>> cloud_;
71 bool binary_;
72 bool use_camera_;
73 K2G & k2g_;
74
75 };
76
77 //entrada por teclado
78 void KeyboardEventOccurred(const pcl::visualization::KeyboardEvent &event, void * data)
79 {
80
81 std::string pressed = event.getKeySym(); // lee los datos de la interrupcion
82 PlySaver * s = (PlySaver*)data;// para guardar la nube de puntos
83 if (event.keyDown())
84 {
85 if (pressed == "a")
86 {
87 if (_useAgc)
88 _useAgc = false;
89 else
90 _useAgc = true;
91 std::cout << "use Agc " << _useAgc << std::endl;
92 }
93 if (pressed == "p")
94 {
95 //se guarda una imagen inicial en cloud2 para luego compararla
96 pcl::copyPointCloud(*cloudAux2, *cloudCopy);
97 tree->setInputCloud(cloudOut);
98 _cluster = true;
99 //_downSample = false;
100 std::cout << "cpiado" << std::endl;
101 }
102 if (pressed == "s")
103 {
104
105 pcl::PLYWriter writer;
106 std::chrono::high_resolution_clock::time_point p = std::chrono::high_resolution_clock::now();
107 std::string now = std::to_string((long)std::chrono::duration_cast<std::chrono::milliseconds>(
108 p.time_since_epoch()).count());
109 writer.write("cloud_" + now, *(s->cloud_), s->binary_, s->use_camera_);
110
111 std::cout << "saved " << "cloud_" + now + ".ply" << std::endl;
112 }
113 if (pressed == "o")
114 {
115 //sirbe para debugear y ver que ocurre en cada ciclo
116 //hay que descomentar una linea en main que hace _ciclo = false
117 if (_ciclo)
118 _ciclo = false;
119 else
120 _ciclo = true;
121
122 std::cout << "ciclo" << std::endl;
123 }
124 if (pressed == "l")
125 {

```

```

126 //sirve para debugear y ver que ocurre en cada ciclo
127 //hay que descomentar una linea en main que hace _ciclo = false
128 _ciclo = true;
129 _ciclo2 = true;
130
131 std::cout << "ciclo" << std::endl;
132 }
133 if (pressed == "u")
134 {
135 if (_resta)
136 _resta = false;
137 else
138 _resta = true;
139 std::cout << "resta " << _resta << std::endl;
140 }
141 if (pressed == "j")
142 {
143 tress = tress + 0.001;
144 std::cout << "tres " << tress << std::endl;
145 }
146 if (pressed == "m")
147 {
148 tress = tress - 0.001;
149 std::cout << "tres " << tress << std::endl;
150 }
151 if (pressed == "y")
152 {
153 if (_downSample)
154 _downSample = false;
155 else
156 _downSample = true;
157 std::cout << "downSample " << _downSample << std::endl;
158 }
159 if (pressed == "h")
160 {
161 down = down + 0.001;
162 std::cout << "down " << down << std::endl;
163 }
164 if (pressed == "n")
165 {
166 down = down - 0.001;
167 std::cout << "down " << down << std::endl;
168 }
169 if (pressed == "x")
170 {
171 s->k2g_.storeParameters();
172 std::cout << "stored calibration parameters" << std::endl;
173 }
174 }
175 }
176
177
178
179 int main(int argc, char * argv[])
180 {
181 //se define e inicia la captura de datos de kinect usando freenect y k2g
182 Processor freenectprocessor =OPENGL;//si no tienes CUDA instalado prueba con CPU, OPENCL o OPENGL
183 std::vector<int> ply_file_indices;
184 K2G k2g(freenectprocessor);
185 k2g.disableLog();
186
187 k2g.printParameters();
188 std::cout << "getting cloud" << std::endl;

```

```

190  cloudKinect = k2g.getCloudMed ();
191
192 //se crea una ventana y se vincula con el objeto viewer
193 boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer(
194             new pcl::visualization::PCLVisualizer("Data Kinect"));
195
196 //se divide la ventana en 2 para mostrar la entrada y la salida
197 //para el lado izq
198 int v1(0);
199 viewer->setPosition(350, 50);
200 viewer->setPointCloudRenderingProperties(pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 300);
201 viewer->createViewPort(0.0, 0.0, 0.5, 1.0, v1);
202 viewer->setBackgroundColor(0, 0, 0, v1);
203 viewer->addPointCloud<PointType>(cloudAux, "sample cloud", v1);
204 viewer->setPointCloudRenderingProperties(
205             pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 5, "sample cloud");
206
207 //para el lado der
208 int v2(0);
209 viewer->createViewPort(0.5, 0.0, 1.0, 1.0, v2);
210 viewer->setBackgroundColor(0.1, 0.1, 0.1, v2);
211 viewer->addText("Filtered data", 10, 10, "v2 text", v2);
212 viewer->addPointCloud<PointType>(cloudOut, "sample cloud2", v2);
213 viewer->setPointCloudRenderingProperties(
214             pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 8, "sample cloud2");
215
216 //se agrega un evento para el teclado para poder guardar la nuve
217 PlySaver ps(cloudAux2, false, false, k2g);
218 viewer->registerKeyboardCallback(KeyboardEventOccurred, (void*)&ps);
219
220 //se acomoda la camara para ver los datos
221 viewer->setCameraPosition(1, 0, -4, 0, -1, 0, 0);
222
223
224 std::ofstream out;
225
226 // std::ios::app is the open mode "append" meaning
227 // new data will be written to the end of the file.
228 out.open("myfile.txt", std::ios::app);
229
230 stringstream myString;
231 myString << std::endl <<
232     "getData\t downsample\t resta\t clusteer\t esfera\t plano\t cilindro\t" <<
233     "veredicto\t probabilidad\t Render\t metodo\t tiempototal"
234     << std::endl;
235 out << myString.str();
236
237 out.close();
238
239 //inicia el ciclo con los datos ya iniciados
240 while (!viewer->wasStopped()) {
241
242
243 out.open("myfile.txt", std::ios::app);
244
245 clock_t timeStart = clock();
246 clock_t timeLoop = clock();
247
248 //se envian los datos de los nuevos puntos y se muestran en la ventana
249 viewer->updatePointCloud(cloudOut, "sample cloud2");
250 viewer->updatePointCloud(cloudAux, "sample cloud");
251
252 //permite la visualizacion de los datos
253 viewer->spinOnce(100, true);

```

```

254
255 //pausa la ejecucion
256 if (_ciclo)
257 {
258 //se obtienen los datos del kinect
259 k2g.updateCloudMed(cloudKinect);
260
261 //se copian a una nube auxiliar
262 copyPointCloud(*cloudKinect, *cloudAux);
263
264 //se limpian las figuras del visor
265 viewer->removeAllShapes();
266 //se re-agrega el menu
267 viewer->addText("Kinect data input", 10, 10, "v1 text", v1);
268 viewer->addText("a: change between ACC an vector metods", 10, 180, "v1 text12", v1);
269 viewer->addText("p: copy to filter", 10, 180, "v1 text3", v1);
270 viewer->addText("s: save cloud to ply", 10, 170, "v1 text2", v1);
271 viewer->addText("o: pausa la ejecucion del programa", 10, 160, "v1 text10", v1);
272 viewer->addText("l: ejecuta un ciclo del programa y se pausa", 10, 150, "v1 text11", v1);
273 viewer->addText("u: on/off filter", 10, 140, "v1 text4", v1);
274 viewer->addText("j: increase threshold filter", 10, 130, "v1 text5", v1);
275 viewer->addText("m: reduce threshold filter", 10, 120, "v1 text6", v1);
276 viewer->addText("y: on/off downn sample", 10, 110, "v1 text7", v1);
277 viewer->addText("h: increase threshold downn sample", 10, 100, "v1 text8", v1);
278 viewer->addText("n: reduce threshold downn sample", 10, 90, "v1 text9", v1);
279
280 if (_debug) {
281 std::cout << "tiempo de actualizacion de los puntos: " <<
282 (double)(clock() - timeStart) / CLOCKS_PER_SEC << std::endl;
283
284 stringstream myString;
285 myString << (double)(clock() - timeStart) / CLOCKS_PER_SEC << "\t";
286 out << myString.str();
287
288 timeStart = clock();
289 }
290 //se realiza una disminucion de puntos
291 if (_downSample)
292 {
293 //vg es el objeto que realiza la disminucion
294 pcl::VoxelGrid<PointType> vg;
295 vg.setInputCloud(cloudAux);
296 vg.setLeafSize(down, down, down);
297 vg.filter(*cloudAux2); // el resultado se guarda en cloudAux2
298
299 }
300 else
301 {
302 copyPointCloud(*cloudAux, *cloudAux2); // el resultado se guarda en cloudAux2
303 }
304
305 if (_debug) {
306 std::cout << "tiempo disminucion de puntos: " <<
307 (double)(clock() - timeStart) / CLOCKS_PER_SEC << std::endl;
308
309 stringstream myString;
310 myString << (double)(clock() - timeStart) / CLOCKS_PER_SEC << "\t";
311 out << myString.str();
312
313 timeStart = clock();
314 }
315 // para realizar la dferencia entre 2 nuves de puntos
316 //para eliminar los puntos que no se usan
317 if (_resta)

```

```

318 {
319 //la resta se realiza cloudAux-cloudCopy con una toleracia tress
320 pcl::SegmentDifferences<PointType> resta;
321 resta.setInputCloud(cloudAux2);
322 resta.setTargetCloud(cloudCopy);
323 resta.setDistanceThreshold(tress);
324 resta.segment(*cloudOut); // la resta se guarda en cloudOut
325 }
326 else
327 {
328 copyPointCloud(*cloudAux2, *cloudOut);
329 }
330
331 if (_debug) {
332 std::cout << "tiempo de resta de las nubes: " <<
333 (double)(clock() - timeStart) / CLOCKS_PER_SEC << std::endl;
334
335 stringstream myString;
336 myString << (double)(clock() - timeStart) / CLOCKS_PER_SEC << "\t";
337 out << myString.str();
338
339 timeStart = clock();
340 }
341
342 //clusterizacion de objetos usando la distancia euclidiana
343 if (_cluster && _resta)
344 {
345 pcl::EuclideanClusterExtraction<PointType> ec;
346 //la toleracia para el cluster es poco mas grande que la de la reduccion
347 ec.setClusterTolerance(down + 0.01);
348 ec.setMinClusterSize(20);
349 ec.setMaxClusterSize(25000);
350 ec.setSearchMethod(tree);
351 ec.setInputCloud(cloudOut);
352
353 //se obtienen los diferentes objetos en diferentes nuves de puntos
354 int j = 0; //contador de clusters
355 //arreglo que contiene los incices de puntos de cada cluster
356 std::vector<pcl::PointIndices> cluster_indices;
357 ec.extract(cluster_indices);
358
359 //para cada nube de puntos(objetos) encontrada:
360 for (std::vector<pcl::PointIndices>::const_iterator it = cluster_indices.begin(); it != cluster_indices.end(); ++it)
361 {
362 //nube de puntos (cluster)
363 pcl::PointCloud<PointType>::Ptr cloud_cluster(new pcl::PointCloud<PointType>);
364 // pcl::PointCloud<PointType>::Ptr cloud_cluster2(new pcl::PointCloud<PointType>);
365 //normales calculadas de la nube de puntos
366 pcl::PointCloud<PointNTType>::Ptr cloud_normals(new pcl::PointCloud<PointNTType>);
367
368 //copia cada punto descrito por los inices
369 for (std::vector<int>::const_iterator pit = it->indices.begin(); pit != it->indices.end(); ++pit)
370 cloud_cluster->push_back(cloudOut->points[*pit]);
371
372 cloud_cluster->width = cloud_cluster->points.size();
373 cloud_cluster->height = 1;
374 cloud_cluster->is_dense = true;
375
376 //se calcula el mejor modelo para cada cluster
377 if (cloud_cluster->width > 20) //el cluster deve tener almenos 20 puntos
378 {
379 double probSphere = 0;
380 double probPlane = 0;

```

```

382 double probCylinder = 0;
383 bool _sphere = false;
384 bool _plane = false;
385 bool _cylinder = false;
386
387 std::vector<int> inliers;//indice de puntos que pertenecen a la figura
388
389
390
391 if (_debug) {
392 std::cout << "tiempo de cluster: " << (double)(clock() - timeStart) / CLOCKS_PER_SEC << std::endl;
393
394 stringstream myString;
395 myString << (double)(clock() - timeStart) / CLOCKS_PER_SEC << "\t";
396 out << myString.str();
397
398 timeStart = clock();
399 }
400 //usando el metodo tradicional
401 if (!useAgc)
402 {
403 /////////////////////////////////modelo de la esfera /////////////////////////////////
404 /////////////////////////////////modelo de la esfera /////////////////////////////////
405 /////////////////////////////////modelo de la esfera /////////////////////////////////
406 pcl::SampleConsensusModelSphere<PointType>::Ptr model_s(
407     new pcl::SampleConsensusModelSphere<PointType>(cloud_cluster));
408 model_s->setRadiusLimits(0.05, 0.5);
409 pcl::RandomSampleConsensus<PointType> ransacS(model_s);
410 // se calcula RANSAC para la esfera
411 ransacS.setDistanceThreshold(.01); // tolerancia de error en la figura
412 ransacS.computeModel();
413 ransacS.getInliers(inliers);
414 probSphere = (double)inliers.size() / cloud_cluster->width;
415
416 if (_debug) {
417 std::cout << "tiempo para estimar la esfera: " <<
418             (double)(clock() - timeStart) / CLOCKS_PER_SEC << std::endl;
419
420 stringstream myString;
421 myString << (double)(clock() - timeStart) / CLOCKS_PER_SEC << "\t";
422 out << myString.str();
423
424 timeStart = clock();
425 }
426
427 /////////////////////////////////modelo del plano /////////////////////////////////
428 /////////////////////////////////modelo del plano /////////////////////////////////
429 /////////////////////////////////modelo del plano /////////////////////////////////
430 pcl::SampleConsensusModelPlane<PointType>::Ptr model_p(
431     new pcl::SampleConsensusModelPlane<PointType>(cloud_cluster));
432 pcl::RandomSampleConsensus<PointType> ransacP(model_p);
433 // se calcula RANSAC para el plano
434 ransacP.setDistanceThreshold(.01); // tolerancia de error en la figura
435 ransacP.computeModel();
436 ransacP.getInliers(inliers);
437 probPlane = (double)inliers.size() / cloud_cluster->width;
438
439 if (_debug) {
440 std::cout << "tiempo para estimar el plano: " <<
441             (double)(clock() - timeStart) / CLOCKS_PER_SEC << std::endl;
442
443 stringstream myString;
444 myString << (double)(clock() - timeStart) / CLOCKS_PER_SEC << "\t";
445 out << myString.str();

```

```

446
447 timeStart = clock();
448 }
449
450 //////////////////////////////////////////////////////////////////// cilindro ///////////////////////////////////////////////////////////////////
451 //////////////////////////////////////////////////////////////////// cilindro ///////////////////////////////////////////////////////////////////
452 //////////////////////////////////////////////////////////////////// cilindro ///////////////////////////////////////////////////////////////////
453 // calculo de normales
454 pcl::NormalEstimation<PointType, PointNType> ne;
455 ne.setSearchMethod(tree);
456 ne.setInputCloud(cloud_cluster);
457 ne.setKSearch(50);
458 ne.compute(*cloud_normals);
459 // modelo del cilindro
460 pcl::SampleConsensusModelCylinder<PointType, PointNType>::Ptr model_c(
461     new pcl::SampleConsensusModelCylinder<PointType, PointNType>(cloud_cluster));
462 model_c->setInputNormals(cloud_normals);
463 model_c->setInputCloud(cloud_cluster);
464 model_c->setRadiusLimits(0.02, 0.8);
465 pcl::RandomSampleConsensus<PointType> ransacC(model_c);
466 // se calcula RANSAC para el cilindro
467 ransacC.setDistanceThreshold(.01); // tolerancia de error en la figura
468 ransacC.computeModel();
469 ransacC.getInliers(inliers);
470 probCylinder = (double)inliers.size() / cloud_cluster->width;
471
472
473 if (_debug) {
474 std::cout << "tiempo para estimar el cilindro: " <<
475 (double)(clock() - timeStart) / CLOCKS_PER_SEC << std::endl;
476
477 stringstream myString;
478 myString << (double)(clock() - timeStart) / CLOCKS_PER_SEC << "\t";
479 out << myString.str();
480
481 timeStart = clock();
482 }
483
484 /*
485 // se muestran las probabilidades para cada objeto
486 std::cout << "sphere prob " << probSphere << std::endl;
487 std::cout << "plane prob " << probPlane << std::endl;
488 std::cout << "cylinder prob " << probCylinder << std::endl;
489 */
490 // se elige la mejor probabilidad
491 if (probSphere > probCylinder && probSphere > probPlane)
492 {
493 stringstream myString;
494 myString << "sphere, \t" << probSphere << "\t";
495 out << myString.str();
496 _sphere = true;
497 }
498 if (probCylinder > probPlane && probCylinder >= probSphere)
499 {
500 stringstream myString;
501 myString << "cylinder, \t" << probCylinder << "\t";
502 out << myString.str();
503 _cylinder = true;
504 }
505 if (probPlane >= probCylinder && probPlane >= probSphere)
506 {
507 stringstream myString;
508 myString << "plane, \t" << probPlane << "\t";
509 out << myString.str();

```

```

510 _plane = true;
511 }
512
513 if (_sphere)
514 {
515 //ransacS.getInliers(inliers);
516
517 //se obtienen los datos de la figura y se convierten en un tipo de dato para graficar
518 Eigen::VectorXf coefficients;
519 ransacS.getModelCoefficients(coefficients);
520 pcl::PointXYZ center;
521 center.x = coefficients[0];
522 center.y = coefficients[1];
523 center.z = coefficients[2];
524
525 //se crea una esfera que corresponde al modelo calculado y se muestra
526 viewer->addSphere(center, coefficients[3], 0.5, 0.0, 0.0, "sphere" + std::to_string(j), o);
527 std::cout << "sphere" + std::to_string(j) << std::endl;
528 _sphere = false;
529 }
530
531 if (_plane)
532 {
533
534 //ransacP.getInliers(inliers); // calcula los puntos de la figura
535 //se obtienen los datos de la figura y se convierten en un tipo de dato para graficar
536 pcl::ModelCoefficients coefficientsM;
537 Eigen::VectorXf coefficients;
538 coefficientsM.values.resize(4);
539 ransacP.getModelCoefficients(coefficients);
540 coefficientsM.values[0] = coefficients[0]; //x
541 coefficientsM.values[1] = coefficients[1]; //y
542 coefficientsM.values[2] = coefficients[2]; //z
543 coefficientsM.values[3] = coefficients[3]; //w//d Hessian component
544 //se crea un plano que corresponde al modelo calculado y se muestra
545 viewer->addPlane(coefficientsM, "plane" + std::to_string(j), o);
546
547 std::cout << "plane" + std::to_string(j) << std::endl;
548 _plane = false;
549 }
550
551 if (_cylinder)
552 {
553 //ransacC.getInliers(inliers);
554 //se obtienen los datos de la figura y se convierten en un tipo de dato para graficar
555 pcl::ModelCoefficients coefficientsM;
556 Eigen::VectorXf coefficients;
557 coefficientsM.values.resize(7);
558 ransacC.getModelCoefficients(coefficients);
559 pcl::PointXYZ center;
560 coefficientsM.values[0] = coefficients[0]; // centro x
561 coefficientsM.values[1] = coefficients[1]; // centro y
562 coefficientsM.values[2] = coefficients[2]; // centro z
563 coefficientsM.values[3] = coefficients[3]; // direccion del eje x
564 coefficientsM.values[4] = coefficients[4]; // direccion del eje y
565 coefficientsM.values[5] = coefficients[5]; // direccion del eje z
566 coefficientsM.values[6] = coefficients[6]; // radio
567 viewer->addCylinder(coefficientsM, "cylinder" + std::to_string(j), o);
568 std::cout << "cylinder" + std::to_string(j) << std::endl;
569 _cylinder = false;
570 }
571
572 if (_debug) {
573

```

```

574 std::cout << "tiempo de renderizacion : " <<
575             (double)(clock() - timeStart) / CLOCKS_PER_SEC << std::endl;
576
577 stringstream myString;
578 myString << (double)(clock() - timeStart) / CLOCKS_PER_SEC << "\t" << _useAgc;
579 out << myString.str();
580
581 timeStart = clock();
582 }
583 }
584 //usando algebra geometrica
585 else
586 {
587
588 if (_debug) {
589 //timeStart = clock();
590 }
591
592 /////////////////////////////////modelo de la esfera /////////////////////////////////
593 /////////////////////////////////se calcula ransac para la esfera ///////////////////////////////
594 /////////////////////////////////se calcula ransac para la esfera en agc
595 ransacSphere sph;
596 sph.setData(cloud_cluster, 00.001);
597 sph.compute();
598 inliers = *sph.indexlist;
599 probSphere = (double)inliers.size() / cloud_cluster->width;
600
601 if (_debug) {
602 std::cout << "tiempo para estimar la esfera: " <<
603                     (double)(clock() - timeStart) / CLOCKS_PER_SEC << std::endl;
604
605 stringstream myString;
606 myString << (double)(clock() - timeStart) / CLOCKS_PER_SEC << "\t";
607 out << myString.str();
608
609 timeStart = clock();
610 }
611
612
613 /////////////////////////////////modelo del plano /////////////////////////////////
614 /////////////////////////////////se calcula ransac para el plano en agc
615 ransacPlane pln;
616 pln.setData(cloud_cluster, 0.01, 1);
617 pln.compute();
618 inliers = *pln.indexlist;
619 probPlane = (double)inliers.size() / cloud_cluster->width;
620
621 if (_debug) {
622 std::cout << "tiempo para estimar el plano: "
623                     << (double)(clock() - timeStart) / CLOCKS_PER_SEC << std::endl;
624
625 stringstream myString;
626 myString << (double)(clock() - timeStart) / CLOCKS_PER_SEC << "\t";
627 out << myString.str();
628
629 timeStart = clock();
630 }
631
632 /////////////////////////////////cilindro /////////////////////////////////
633 /////////////////////////////////cilindro /////////////////////////////////
634 /////////////////////////////////cilindro /////////////////////////////////
635 /////////////////////////////////cilindro /////////////////////////////////
636 /////////////////////////////////cilindro /////////////////////////////////
637 /////////////////////////////////cilindro /////////////////////////////////

```

```

638 //se calcula ransac para el cilindro en agc
639 ransacCylinder2 cyl;
640 cyl.setData(cloud_cluster, 0.02);
641 cyl.compute();
642 inliers = *cyl.indexlist;
643 probCylinder = (double)inliers.size() / cloud_cluster->width;
644
645 if (_debug) {
646 std::cout << "tiempo para estimar el cilindro: " <<
647 (double)(clock() - timeStart) / CLOCKS_PER_SEC << std::endl;
648
649 stringstream myString;
650 myString << (double)(clock() - timeStart) / CLOCKS_PER_SEC << "\t";
651 out << myString.str();
652
653 timeStart = clock();
654 }
655
656
657 //se muestran las probabilidades para cada objeto
658 std::cout << "sphere prob \t" << probSphere << std::endl;
659 std::cout << "plane prob \t" << probPlane << std::endl;
660 std::cout << "cylinder prob \t" << probCylinder << std::endl;
661
662 //se elige la mejor probabilidad
663 if (probSphere > probCylinder && probSphere > probPlane)
664 {
665 stringstream myString;
666 myString << "sphere\t" << probSphere << "\t";
667 out << myString.str();
668 _sphere = true;
669 }
670 if (probCylinder > probPlane && probCylinder >= probSphere)
671 {
672 stringstream myString;
673 myString << "cylinder\t" << probCylinder << "\t";
674 out << myString.str();
675 _cylinder = true;
676 }
677 if (probPlane >= probCylinder && probPlane >= probSphere)
678 {
679 stringstream myString;
680 myString << "plane\t" << probPlane << "\t";
681 out << myString.str();
682 _plane = true;
683 }
684
685 if (_sphere) {
686 //inliers = *sph.indexlist;
687 pcl::PointXYZ center;
688 center.x = sph.ranSph[0];
689 center.y = sph.ranSph[1];
690 center.z = sph.ranSph[2];
691 //se crea una esfera que corresponde al modelo calculado y se muestra
692 viewer->addSphere(center, sph.rad, 0.5, 0.0, 0.0, "sphere" + std::to_string(j), 0);
693
694 std::cout << "sphere" + std::to_string(j) << std::endl;
695 _sphere = false;
696
697 }
698
699 if (_plane)
700 {
701 //inliers = *pln.indexlist;

```

```

702 pcl::PointXYZ center;
703 pcl::ModelCoefficients coefficientsM;
704 coefficientsM.values.resize(4);
705 coefficientsM.values[0] = pln.ranPln[0];//x normal
706 coefficientsM.values[1] = pln.ranPln[1];//y normal
707 coefficientsM.values[2] = pln.ranPln[2];//z normal
708 coefficientsM.values[3] = -1 * pln.ranPln[4];//d Hessian component
709
710 //se crea una esfera que corresponde al modelo calculado y se muestra
711 viewer->addPlane(coefficientsM, "plane" + std::to_string(j), o);
712
713 std::cout << "plane" + std::to_string(j) << std::endl;
714 _plane = false;
715 }
716
717 if (_cylinder)
718 {
719
720 //inliers = *cyl.indexlist;
721 //se obtienen los datos de la figura y se convierten en un tipo de dato para graficar
722 pcl::ModelCoefficients coefficientsM;
723 Eigen::VectorXf coefficients;
724 coefficientsM.values.resize(7);
725 pcl::PointXYZ center;
726 coefficientsM.values[0] = cyl.ranCyl.center[0];// centro x
727 coefficientsM.values[1] = cyl.ranCyl.center[1];// centro y
728 coefficientsM.values[2] = cyl.ranCyl.center[2];// centro z
729 coefficientsM.values[3] = cyl.ranCyl.plan[0]*2;// direccion del eje x
730 coefficientsM.values[4] = cyl.ranCyl.plan[1]*2;// direccion del eje y
731 coefficientsM.values[5] = cyl.ranCyl.plan[2]*2;// direccion del eje z
732 coefficientsM.values[6] = cyl.ranCyl.radius;// radio
733 viewer->addCylinder(coefficientsM, "cylinder" + std::to_string(j), o);
734
735 pcl::PointXYZ center1;
736 center1.x = cyl.ranCyl.s1.x;
737 center1.y = cyl.ranCyl.s1.y;
738 center1.z = cyl.ranCyl.s1.z;
739 //se crea una esfera que corresponde al modelo calculado y se muestra
740 viewer->addSphere(center1, cyl.ranCyl.s1.radius, 0.5, 0.0, 0.0, "sphere1" + std::to_string(j), o);
741
742 pcl::PointXYZ center2;
743 center2.x = cyl.ranCyl.s2.x;
744 center2.y = cyl.ranCyl.s2.y;
745 center2.z = cyl.ranCyl.s2.z;
746 //se crea una esfera que corresponde al modelo calculado y se muestra
747 viewer->addSphere(center2, cyl.ranCyl.s2.radius, 0.5, 0.0, 0.0, "sphere2" + std::to_string(j), o);
748
749 std::cout << "cylinder" + std::to_string(j) << std::endl;
750 _cylinder = false;
751 }
752
753 //se copian solo los puntos de la figura(esfera, plano o cilindro) a otra nube de puntos
754 // pcl::copyPointCloud<PointType>(*cloud_cluster, inliers, *cloud_cluster2);
755 //se agrega la nube resultante a la ventana para la visualizacion
756 // viewer2->addPointCloud<PointType>(cloud_cluster2, "cloud" + std::to_string(j));
757 //se suma uno al contador de cluster
758 j++;
759 // std::cout << "cluster" + std::to_string(j) << std::endl;
760
761 if (_debug) {
762
763 std::cout << "tiempo de renderizacion : " <<
764 (double)(clock() - timeStart) / CLOCKS_PER_SEC << std::endl;
765

```

```

766 stringstream myString;
767 myString << (double)(clock() - timeStart) / CLOCKS_PER_SEC << "\t" << _useAgc;
768 out << myString.str();
769
770 timeStart = clock();
771 }
772 }
773 }
774 }
775 }
776 }
777 else {
778
779
780 stringstream myString;
781 myString << ", \t, \t, \t, \t, \t, \t" ;
782 out << myString.str();
783 }
784 stringstream myString;
785 myString << ", \t" << (double)(clock() - timeLoop) / CLOCKS_PER_SEC << std::endl;
786 out << myString.str();
787
788 // si se usa la tecla de ciclo unico
789 if (_ciclo2)
790 _ciclo = false;
791
792 out.close();
793 }
794 }
795
796 k2g.shutDown();
797 return 0;
798 }

```

Código B.8: Archivo ransac.h

```

1  //
2  // Created by Aksel Sveier. Spring 2016
3  //
4
5  #ifndef RANSACGA_RANSACH
6  #define RANSACGA_RANSACH
7
8  #endif //RANSACGA_RANSACH
9
10 #include <space/vsr_cg3D_op.h>
11 #include <pcl/point_types.h>
12 #include <pcl/common/common_headers.h>
13 #include <pcl/filters/passthrough.h>
14 #include <pcl/octree/octree.h>
15 #include <pcl/octree/octree_impl.h>
16 #include <pcl/filters/extract_indices.h>
17
18 #include <cmath>           // std::abs
19 #include "objects.h"
20
21
22 using namespace vsr;
23 using namespace vsr::cga;
24 using namespace pcl;

```

```

25 //classes that performs ransac operations depending on the object(line, plane, sphere)
26
27
28
29 ///////////////////////////////////////////////////////////////////
30 /////////////////////////////////////////////////////////////////// SPHERE ///////////////////////////////////////////////////////////////////
31 ///////////////////////////////////////////////////////////////////
32
33
34
35 //Class for detecting a sphere in a point cloud with ransac
36 class ransacSphere{
37 public:
38 //Class constructor
39 ransacSphere() {
40 indexlist = new std::vector<int>;
41 }
42
43 float rad=0;
44 float radiusTolerance=0.001;
45 //For the kinect the accuracy is 1 % of the distance from the sensor for 100 images.
46 float inlierTreshold = 0.005;
47 float maxRadius = 1;
48 int candidates = 1; //Number of candidates to detect
49 int iterations = 1000; //Maximum allowed iterations
50 int actualIt; //Integer for storing the number of iterations performed
51 int numInliers; //Integer for storing the number of inliers
52 std::vector<int> *indexlist; //Vector for storing inliers
53 PointCloud<PointXYZRGB>::Ptr cloud; //Point cloud for storing the data set
54 DualSphere ranSph; //Sphere object
55 sphere asphere; //Create a sphere object from object.h
56
57 sphere sph1;//para el calculo del cilindro
58 float distS1S2 = 0; //distancia entre los centros de s1 y s2
59 Pnt s1;
60
61
62 //Setting the data and parameters of the algoritm
63 void setData(PointCloud<PointXYZRGB>::Ptr cl , float tol) {
64 cloud = cl;
65 //radius = rad;
66 inlierTreshold = tol;
67 candidates = 1;
68 }
69
70 //Setting the data and parameters of the algoritm
71 void setData(PointCloud<PointXYZRGB>::Ptr cl , float tol , int iter) {
72 cloud = cl;
73 iterations = iter;
74 inlierTreshold = tol;
75 candidates = 1;
76 }
77 //Setting the data and parameters of the algoritm
78 void setData(PointCloud<PointXYZRGB>::Ptr cl , float tol , int iter , sphere sp , float dist) {
79 cloud = cl;
80 iterations = iter;
81 inlierTreshold = tol;
82 candidates = 1;
83 sph1 = sp;
84 s1 = Vec(sph1.x, sph1.y, sph1.z).null();
85 distS1S2 = dist;
86
87 }
88 /*

```

```

89 //Setting the data and parameters of the algorithm
90 void setData(PointCloud<PointXYZRGB>::Ptr cl, float tol, int cand) {
91   cloud = cl;
92   //radius = rad;
93   inlierTreshold = tol;
94   candidates = cand;
95 }
96 */
97 //Runs the primitive shape detection for a sphere
98 bool compute() {
99   //Variables to keep track of the algorithm
100  int it = 0;
101  vector<int> ran(4);
102  int can = 0;
103  vector<sphere> cand(candidates);
104  int count;
105  vector<int> inPoints(candidates);
106
107  //Timer
108  std::clock_t t1, t2;
109  t1 = std::clock();
110
111
112  ///vars for probability stop
113  double k = 3.0;
114  double probability_ = 0.99;//prob. to get a selection without error
115  double log_probability = log(1.0 - probability_);
116  double one_over_indices = 1.0 / static_cast<double>(cloud->width);
117
118
119  inPoints[0] = 0;
120
121  //Algorithm
122  //while (it < iterations) { //can < candidates && it < iterations) {
123  while (it < k && it < iterations) {
124
125    //Generate random indexes
126    for (int i = 0; i < 4; i++) {
127      ran[i] = rand() % cloud->points.size();
128    }
129
130    //Creates a dual sphere in conformal space from 4 indexed points in a point cloud
131    asphere.defineDual(cloud, ran);
132
133    //Check if sphere is inside tolerance
134    if (sphereCheck(asphere)) {
135
136      //Count inliers
137      count = 0;
138      for (int j = 0; j < cloud->points.size(); j++) {
139        if (isInlier(Vec(cloud->points[j].x, cloud->points[j].y, cloud->points[j].z).null(), asphere)) {
140          count++;
141        }
142      }
143      /*
144      if (inPoints[0] == NULL) {
145        inPoints[0] = count;
146        cand[0] = asphere;
147      }
148      */
149      if (count > inPoints[0]) {
150        cand[0] = asphere;
151        inPoints[0] = count;
152        can++;

```

```

153
154 // Compute the k parameter (k=log(z)/log(1-w^n))
155 double w = static_cast<double> (inPoints[o]) * one_over_indices;
156 double p_no_outliers = 1.0 - pow(w, (ran.size())); // 4 puntos para crear la esfera
157 // Avoid division by -Inf
158 p_no_outliers = (std::max) (std::numeric_limits<double>::epsilon(), p_no_outliers);
159 // Avoid division by 0.
160 p_no_outliers = (std::min) (1.0 - std::numeric_limits<double>::epsilon(), p_no_outliers);
161 k = log_probability / log(p_no_outliers);
162
163 }
164 }
165 it++;
166
167
168
169 }
170 }
171
172 if (can > o) {
173 //Find candidate with most inliers
174 int best = o;
175 numInliers = o;
176 for (int i = o; i < cand.size(); i++) {
177 if (inPoints[i] > numInliers) {
178 best = i;
179 numInliers = inPoints[i];
180 }
181 }
182
183 //cand[best].radius = sqrt(-2 * cand[best].dualSphere[4]);
184 asphere = cand[best];
185 ranSph = cand[best].dualSphere;
186 rad = cand[best].calcRadius(cand[best].dualSphere);
187
188
189 delete indexlist;
190 indexlist = new std::vector<int>;
191 for (int j = o; j < cloud->points.size(); j++) {
192 if (isInlier(Vec(cloud->points[j].x, cloud->points[j].y, cloud->points[j].z).null(), asphere)) {
193 indexlist->push_back(j);
194 }
195 }
196
197
198
199 //Check if the fit is succesfull, return the unfitted sphere.
200 /*if (cand[best].radius / rad > 1.1 || cand[best].radius / rad < 0.9) {
201 ranSph = cand[best].dualSphere;
202 }*/
203 //Return true if a sphere was found
204 return true;
205 }
206 else {
207 //Return false if not
208 return false;
209 }
210 }
211
212 //Runs the primitive shape detection for a sphere2 of the cylinder
213 bool computeS2() {
214 //Variables to keep track of the algorithm
215 int it = o;
216 vector<int> ran(4);

```

```

217     int can = o;
218     vector<sphere> cand(candidates);
219     int count;
220     vector<int> inPoints(candidates);
221     Pnt s2;
222
223     //Timer
224     std::clock_t t1, t2;
225     t1 = std::clock();
226
227
228     ///vars for probability stop
229     double k = 3.0;
230     double probability_ = 0.99;//prob. to get a selection without error
231     double log_probability = log(1.0 - probability_);
232     double one_over_indices = 1.0 / static_cast<double>(cloud->width);
233
234
235     inPoints[o] = o;
236
237     //Algorithm
238     //while (it < iterations) { //can < candidates && it < iterations) {
239     while (it < k && it < iterations) {
240
241         //Generate random indexes
242         for (int i = o; i < 4; i++) {
243             ran[i] = rand() % cloud->points.size();
244         }
245
246         //Creates a dual sphere in conformal space from 4 indexed points in a point cloud
247         asphere.defineDual(cloud, ran);
248
249         s2 = Vec(asphere.x, asphere.y, asphere.z).null();
250
251         Sca d = Sca(-2)*(s1 <= s2);
252
253         //Check if sphere is inside tolerance
254         if (sphereCheck(asphere) && distS1S2 < sqrt ( abs ( d[o] ) ) ) {
255
256             //Count inliers
257             count = o;
258             for (int j = o; j < cloud->points.size(); j++) {
259                 if (isInlier(Vec(cloud->points[j].x, cloud->points[j].y, cloud->points[j].z).null(), asphere)) {
260                     count++;
261                 }
262             }
263             /*
264             if (inPoints[o] == NULL) {
265                 inPoints[o] = count;
266                 cand[o] = asphere;
267             }
268             */
269             if (count > inPoints[o]) {
270                 cand[can] = asphere;
271                 inPoints[can] = count;
272                 can++;
273
274             // Compute the k parameter (k=log(z)/log(1-w^n))
275             double w = static_cast<double>(inPoints[o]) * one_over_indices;
276             double p_no_outliers = 1.0 - pow(w, (ran.size()));// 4 puntos para crear la esfera
277             // Avoid division by -Inf
278             p_no_outliers = (std::max)(std::numeric_limits<double>::epsilon(), p_no_outliers);
279             // Avoid division by 0.
280             p_no_outliers = (std::min)(1.0 - std::numeric_limits<double>::epsilon(), p_no_outliers);

```

```

281     k = log_probability / log(p_no_outliers);
282
283     }
284     }
285     it++;
286
287
288
289     }
290
291     if (can > o) {
292         //Find candidate with most inliers
293         int best = o;
294         numInliers = o;
295         for (int i = o; i < cand.size(); i++) {
296             if (inPoints[i] > numInliers) {
297                 best = i;
298                 numInliers = inPoints[i];
299             }
300         }
301     }
302
303     //cand[best].radius = sqrt(-2 * cand[best].dualSphere[4]);
304     asphere = cand[best];
305     ranSph = cand[best].dualSphere;
306     rad = cand[best].calcRadius(cand[best].dualSphere);
307
308     delete indexlist;
309     indexlist = new std::vector<int>;
310     for (int j = o; j < cloud->points.size(); j++) {
311         if (isInlier(Vec(cloud->points[j].x, cloud->points[j].y, cloud->points[j].z).null(), asphere)) {
312             indexlist->push_back(j);
313         }
314     }
315 }
316
317
318
319     //Check if the fit is succesfull, return the unfitted sphere.
320     /*if (cand[best].radius / rad > 1.1 || cand[best].radius / rad < 0.9) {
321         ranSph = cand[best].dualSphere;
322     }*/
323     //Return true if a sphere was found
324     return true;
325 }
326 else {
327     //Return false if not
328     return false;
329 }
330 }

331
332
333     private:
334
335     //Function to check if a point is classified as a inlier
336     bool isInlier(Pnt point, sphere can) {
337         Sca dist = Sca(2)*(point <= can.dualSphere) ; // z (P . S*)
338         if ((abs(dist[o]) < inlierThreshold) && (abs(dist[o]) > -1* inlierThreshold)) {
339             return true;
340         }
341         else {
342             return false;
343         }
344     }

```

```

345
346 //Check if the radius is inside the tolerance limits
347 bool sphereCheck(sphere sph) {
348     if (sph.radius < maxRadius)
349     {
350         return true;
351     }
352     else {
353         return false;
354     }
355 }
356 };
357
358 ///////////////////////////////////////////////////////////////////
359 ///////////////////////////////////////////////////////////////////      PLANE      ///////////////////////////////////////////////////////////////////
360 ///////////////////////////////////////////////////////////////////
361 ///////////////////////////////////////////////////////////////////
362 ///////////////////////////////////////////////////////////////////
363
364 //Class for detecting a plane in a point cloud with RANSAC
365 class ransacPlane{
366 public:
367     //Constructor
368     ransacPlane() {
369         indexlist = new std::vector<int>;
370     }
371     float planetol; //Tolerance used for deciding whether a point is compatible with a plane
372     PointCloud<PointXYZRGB>::Ptr cloud; //Point cloud for holding the data set
373     PointCloud<PointXYZRGB>::Ptr segment; //Point cloud for holding the inlier points
374     std::vector<int> *indexlist; //Vector for holding the index of inlier points
375
376     int iterations = 1000; //Maximum allowed iterations
377     Pnt ranPln; //Plane object represented in IPNS
378     int candidates; //Number of candidates to detect
379
380
381
382
383
384     //Setting the data and parameters of the algorithm
385     void setData(PointCloud<PointXYZRGB>::Ptr cl, float tol, int cand) {
386         cloud = cl;
387         planetol = tol;
388         candidates = cand;
389     }
390
391     //Runs the primitive shape detection for a plane
392     bool compute() {
393         //Variables to keep track of the algorithm
394         vector<int> ran(3);
395         int count;
396         vector<Pnt> cand(candidates);
397         plane aplane; //Create a plane object from object.h
398         int can = 0;
399         int it = 0;
400         vector<int> inPoints(candidates);
401
402
403         //Vars for probability stop
404         double k = 3.0;
405         double probability_ = 0.99;//prob. to get a selection without error
406         double log_probability = log(1.0 - probability_);
407         double one_over_indices = 1.0 / static_cast<double> (cloud->width);
408

```

```

409     inPoints[o] = o;
410     //Algorithm
411     //while (it < iterations) {
412     while (it < k && it < iterations) {
413
414         //Generate random indexesx
415         for (int i = o; i<3; i++) {
416             ran[i] = rand() % cloud->points.size();
417         }
418         //Create dual plane with indexed points from point cloud, using GA
419         //Creates a dual plane in conformal space from 3 indexed points in a point cloud
420         aplane.defineDual(cloud, ran);
421         cand[o] = aplane.normDualPlane;
422
423         //Find number of inlier points for each candidate
424         count = o;
425         for (int j = o; j < cloud->points.size(); j++) {
426             if (isInlier(Vec(cloud->points[j].x, cloud->points[j].y, cloud->points[j].z).null(), cand[o])) {
427                 count++;
428             }
429         }
430         /*
431         if (inPoints[o] == NULL) {
432             inPoints[o] = count;
433         }
434         */
435         if (count > inPoints[o]) {
436             cand[o] = aplane.normDualPlane;
437             inPoints[o] = count;
438         }
439         can++;
440
441         // Compute the k parameter (k=log(z)/log(1-w^n))
442         double w = static_cast<double> (inPoints[o]) * one_over_indices;
443         double p_no_outliers = 1.0 - pow(w, (ran.size())); // 4 puntos para crear la esfera
444         // Avoid division by -Inf
445         p_no_outliers = (std::max) (std::numeric_limits<double>::epsilon(), p_no_outliers);
446         // Avoid division by 0.
447         p_no_outliers = (std::min) (1.0 - std::numeric_limits<double>::epsilon(), p_no_outliers);
448         k = log_probability / log(p_no_outliers);
449
450     }
451
452     it++;
453 }
454
455     if (can > o) {
456         //Find candidate with most inliers
457         int best = o;
458         int numBest = o;
459         for (int i = o; i < cand.size(); i++) {
460             if (inPoints[i] > numBest) {
461                 best = i;
462                 numBest = inPoints[i];
463             }
464         }
465
466         delete indexlist;
467         indexlist = new std::vector<int>;
468         for (int j = o; j < cloud->points.size(); j++) {
469             if (isInlier(Vec(cloud->points[j].x, cloud->points[j].y, cloud->points[j].z).null(), cand[o])) {
470                 indexlist->push_back(j);
471             }
472         }

```

```

473     ranPln = cand[o];
474     return true;
475   }
476   else {
477     return false;
478   }
479 }
480 }
481 }
482 }
483 private:
484
485 //Function to check if a point is classified as a inlier
486 bool isInlier(Pnt point, Pnt plane) {
487   Sca distance = point <= plane;
488   if (abs(distance[o]) < planetol) {
489     return true;
490   }
491   else {
492     return false;
493   }
494 }
495 }
496 };
497
498
499 ///////////////////////////////////////////////////////////////////
500 /////////////////////////////////////////////////////////////////// CYLINDER ///////////////////////////////////////////////////////////////////
501
502 ///////////////////////////////////////////////////////////////////
503
504
505 //Class for detecting a cylinder in a point cloud using the sphere–sphere approach.
506 class ransacCylinder2{
507 public:
508 //Constructor
509 ransacCylinder2() {
510   indexlist = new std::vector<int>;
511 }
512 PointCloud<PointXYZRGB>::Ptr cloud; //Point cloud for storing the data
513 PointCloud<PointXYZRGB>::Ptr inlierCloud; //Point cloud for storing the inliers
514 float tol = o; //Float for the radius tolerance
515 float searchRadius = o; //Float for storing the radius
516 int inliers; //Integer for storing the number of inliers
517 cylinder ranCyl; //Cylinder object
518 ransacSphere sph; //Sphere detection object
519 std::vector<int> *indexlist; //Vector for holding the index of inlier points
520 int candidates = 1;
521 int it=o;
522 int iterations = 3;
523 int count;
524 std::vector<int> ran;
525 int can = o;
526 int numInliers;
527
528
529
530 //Setting the data and parameters of the algorithm
531 void setData(PointCloud<PointXYZRGB>::Ptr cl, float tolerance) {
532   cloud = cl;
533   tol = tolerance;
534   //searchRadius = rad;
535 }
536

```

```

537 //Runs the primitive shape detection for a cylinder using the sphere–sphere approach
538 bool compute() {
539
540     vector<int> ran(4);
541     vector<cylinder> cand(candidates);
542     vector<int> inPoints(candidates);
543     //Variables to keep track of the algorithm
544     inlierCloud.reset(new PointCloud<PointXYZRGB>);
545     int icount=0;
546     Pnt p1, p2;
547     float pirad, p2rad;
548     sphere s1, s2;
549     ransacSphere sph;
550
551
552     ///vars for probability stop
553     double k = 300.0;
554     double probability_ = 0.9;//prob. to get a selection without error
555     double log-probability = log(1.0 - probability_);
556     double one_over_indices = 1.0 / static_cast<double>(cloud->width);
557
558
559     Pnt ni = Vec().null();
560     ni[0] = 0;
561     ni[1] = 0;
562     ni[2] = 0;
563     ni[3] = 0;
564     ni[4] = 1;
565
566     inPoints[0] = 0;
567     //Algorithm
568     //while (it < iterations) {
569     while (it < k && it < iterations) {
570
571
572
573
574
575     //se calcula ransac para la esfera en agc
576     sph.setData(cloud, 0.001);
577     sph.compute();
578     s1 = sph.asphere;
579
580     //se calcula ransac para la esfera en agc
581     sph.setData(cloud, 0.001);
582     sph.compute();
583     s2 = sph.asphere;
584
585
586     searchRadius = (s1.radius + s2.radius) / 2;
587
588     Par ppar = s1.dualSphere ^ s2.dualSphere;
589     Lin L1 = ppar ^ ni;
590
591     //Find number of inlier points for each candidate
592     count = 0;
593     if (searchRadius < 1.0) {
594         for (int j = 0; j < cloud->points.size(); j++) {
595             if (isInlier2(cloud->points[j].x, cloud->points[j].y,
596                           cloud->points[j].z, L1, searchRadius)) {
597                 count++;
598             }
599         }
600     }

```

```

601
602     }
603     /*
604     if (inPoints[o] == NULL) {
605         cand[o] = ranCyl;
606         inPoints[o] = count;
607         can++;
608     }
609     */
610     if (count > inPoints[o]) {
611         cand[o].defineCylinder(s1, s2, searchRadius);
612         inPoints[o] = count;
613         can++;
614
615     // Compute the k parameter (k=log(z)/log(1-w^n))
616     double w = static_cast<double>(inPoints[o]) * one_over_indices;
617     double p_no_outliers = 1.0 - pow(w, (8)); // 4 puntos para crear la esfera 8 para el cilindro
618     // Avoid division by -Inf
619     p_no_outliers = (std::max)(std::numeric_limits<double>::epsilon(), p_no_outliers);
620     // Avoid division by 0.
621     p_no_outliers = (std::min)(1.0 - std::numeric_limits<double>::epsilon(), p_no_outliers);
622     k = log_probability / log(p_no_outliers);
623
624
625     }
626
627     it++;
628 }
629
630
631
632
633
634     if (can > o ) {
635         //Find candidate with most inliers
636         int best = o;
637         numInliers = o;
638         for (int i = o; i < cand.size(); i++) {
639             if (inPoints[i] > numInliers) {
640                 best = i;
641                 numInliers = inPoints[i];
642             }
643         }
644
645     ranCyl = cand[best];
646
647
648
649     Lin L1 = ranCyl.s1.dualSphere ^ ranCyl.s2.dualSphere ^ ni;
650     delete indexlist;
651     indexlist = new std::vector<int>;
652     if (ranCyl.radius < 1) {
653         for (int j = o; j < cloud->points.size(); j++) {
654             if (isInlier2(cloud->points[j].x, cloud->points[j].y, cloud->points[j].z, L1, ranCyl.radius)) {
655                 indexlist->push_back(j);
656             }
657         }
658     }
659
660
661     return true;
662 }
663 else {
664     //Return false if not

```

```

665     return false;
666 }
667
668
669
670
671 }
672
673
674
675     private:
676     //Function to check if a point is classified as a inlier
677     bool isInlier2(float x, float y, float z, Lin L, float radius) {
678         Pnt p1 = Vec(x, y, z).null();
679         float rMax, rMin;
680
681         rMax = radius + tol;
682         rMin = radius - tol;
683
684         DualSphere s= ((p1 ^ L.dual()) / L.dual());
685         Sca d = ( s <= s);
686         if ( (sqrt( abs(d[o])) > (rMin)) && (sqrt(abs(d[o])) < (rMax))) {
687             return true;
688         }
689         else {
690             return false;
691         }
692     }
693 }
694
695 };

```

Código B.9: Archivo objects.h

```

1 //
2 // Created by Aksel Sveier. Spring 2016
3 //
4
5 #ifndef RANSACGA_OBJECTS_H
6 #define RANSACGA_OBJECTS_H
7
8 #endif //RANSACGA_OBJECTS_H
9
10 #include <space/vsr_cga3D_op.h>
11 #include <pcl/point_types.h>
12 #include <pcl/common/common_headers.h>
13
14 using namespace vsr;
15 using namespace vsr::cga;
16 using namespace pcl;
17
18
19 //Class that creates and handles spheres with CGA
20 class sphere{
21
22 public:
23     float radius, x, y, z; //Floats for storing the parameters of the sphere
24     Pnt dualSphere; //Stores the IPNS representation
25     Sph sphe; //Stores the OPNS representation
26

```

```

27 //Creates a sphere from center(x,y,z) coordinates and radius
28 void defineDual(float x, float y, float z, float r) {
29 dualSphere = normalize(Round::dls(Vec(x, y, z), r));
30 radius = r;
31 x = dualSphere[0]; y = dualSphere[1]; z = dualSphere[2];
32 }
33
34 //Creates a sphere from 4 points in conformal space.
35 void defineDual(Pnt p1, Pnt p2, Pnt p3, Pnt p4) {
36 dualSphere = normalize((p1^p2^p3^p4).dual());
37 radius = calcRadius(dualSphere);
38 x = dualSphere[0]; y = dualSphere[1]; z = dualSphere[2];
39 }
40
41 //Creates a sphere from 4 points from a pointcloud.
42 void defineDual(PointCloud<PointXYZRGB>::Ptr cl1,
43                 PointCloud<PointXYZRGB>::Ptr cl2,
44                 PointCloud<PointXYZRGB>::Ptr cl3,
45                 PointCloud<PointXYZRGB>::Ptr cl4) {
46 dualSphere = normalize((Vec(cl1->points[0].x, cl1->points[0].y, cl1->points[0].z).null()
47 ^ Vec(cl2->points[0].x, cl2->points[0].y, cl2->points[0].z).null()
48 ^ Vec(cl3->points[0].x, cl3->points[0].y, cl3->points[0].z).null()
49 ^ Vec(cl4->points[0].x, cl4->points[0].y, cl4->points[0].z).null()).dual());
50 radius = calcRadius(dualSphere);
51 x = dualSphere[0]; y = dualSphere[1]; z = dualSphere[2];
52 }
53
54 //Creates a sphere in from 4 indexed points in a point cloud
55 void defineDual(PointCloud<PointXYZRGB>::Ptr cl, vector<int> index) {
56 dualSphere = normalize(
57     (Vec(cl->points[index[0]].x, cl->points[index[0]].y, cl->points[index[0]].z).null()
58 ^ Vec(cl->points[index[1]].x, cl->points[index[1]].y, cl->points[index[1]].z).null()
59 ^ Vec(cl->points[index[2]].x, cl->points[index[2]].y, cl->points[index[2]].z).null()
60 ^ Vec(cl->points[index[3]].x, cl->points[index[3]].y, cl->points[index[3]].z).null()).dual());
61
62 radius = calcRadius(dualSphere);
63 x = dualSphere[0]; y = dualSphere[1]; z = dualSphere[2];
64
65 }
66 //Function that calculates radius
67 float calcRadius(Pnt sph) {
68 //return sqrt((1 / pow(sph[3], 2))*(pow(sph[0], 2) +
69 //           pow(sph[1], 2) + pow(sph[2], 2)) - (2 * sph[4] / sph[3]));
70 Sca rad = Sca(2) * (Vec(sph[0], sph[1], sph[2]).null() <= sph);
71 return sqrt(rad[0]);
72 }
73
74 private:
75 //Function that normalizes a conformal vector
76 Pnt normalize(Pnt dSph) {
77 Pnt ret;
78 for (int i = 0; i < 5; i++) {
79 ret[i] = dSph[i] / dSph[3];
80 }
81 return ret;
82 }
83
84 };
85
86 //Class that creates and handles planes with CGA
87 class plane
88 {
89
90

```

```

91 public:
92 Pnt dualPlane; //Stores the IPNS representation
93 Pnt normDualPlane; //Stores the normalized IPNS representation
94
95
96 //Creates a plane from 3 points in conformal space
97 void defineDual(Pnt p1, Pnt p2, Pnt p3) {
98 dualPlane = (p1^p2^p3^Inf(1)).dual();
99 normalize(dualPlane);
100 }
101
102 //Creates a plane from 3 indexed points in a point cloud
103 void defineDual(PointCloud<PointXYZRGB>::Ptr cl, vector<int> index) {
104 dualPlane = (Vec(cl->points[index[0]].x, cl->points[index[0]].y, cl->points[index[0]].z).null()
105 ^ Vec(cl->points[index[1]].x, cl->points[index[1]].y, cl->points[index[1]].z).null()
106 ^ Vec(cl->points[index[2]].x, cl->points[index[2]].y, cl->points[index[2]].z).null()
107 ^ Inf(1)).dual();
108 normalize(dualPlane);
109 }
110 private:
111 //Function that normalizes a conformal vector
112 void normalize(Pnt dPln) {
113 for (int i = 0; i < 5; i++) {
114 normDualPlane[i] = dPln[i] / sqrt(pow(dPln[0], 2) + pow(dPln[1], 2) + pow(dPln[2], 2));
115 }
116 }
117
118
119 };
120
121 //Class that creates and handles circles with CCA
122 class circle
123 {
124
125 public:
126 float radius = 0; //Float for storing the circle radius
127 Pnt circleCenter; //Point in conformal space for storing the circle center
128 Par circ; //Store the OPNS representation
129 Cir dualCircle; //Stores the IPNS representation
130 Pnt plane; //Stores the plane the the circle lie on
131
132
133 //Creates circle from 3 points in conformal space
134 void defineCircle(Pnt p1, Pnt p2, Pnt p3) {
135 dualCircle = p1 ^ p2 ^ p3;
136 circ = (p1 ^ p2 ^ p3).dual();
137 Pnt temp = ((p1 ^ p2 ^ p3).dual()) * Inf(1) * ((p1 ^ p2 ^ p3).dual());
138 for (int i = 0; i < 5; i++) {
139 circleCenter[i] = temp[i] / temp[3];
140 }
141 findNormal((p1 ^ p2 ^ p3 ^ Inf(1)).dual());
142 calcRadius(p1);
143 }
144
145 //Creates a circle from 3 point from a point cloud
146 void defineCircle(PointCloud<PointXYZRGB>::Ptr cl, vector<int> index) {
147 dualCircle = Vec(cl->points[index[0]].x, cl->points[index[0]].y, cl->points[index[0]].z).null()
148 ^ Vec(cl->points[index[1]].x, cl->points[index[1]].y, cl->points[index[1]].z).null()
149 ^ Vec(cl->points[index[2]].x, cl->points[index[2]].y, cl->points[index[2]].z).null();
150 circ = dualCircle.dual();
151 Pnt temp = circ * Inf(1) * circ;
152 for (int i = 0; i < 5; i++) {
153 circleCenter[i] = temp[i] / temp[3];
154 }

```

```

155 findNormal((Vec(cl->points[index[0]].x, cl->points[index[0]].y, cl->points[index[0]].z).null() ^
156     ^ Vec(cl->points[index[1]].x, cl->points[index[1]].y, cl->points[index[1]].z).null() ^
157     ^ Vec(cl->points[index[2]].x, cl->points[index[2]].y, cl->points[index[2]].z).null() ^
158     ^ Inf(1)).dual());
159 calcRadius(Vec(cl->points[index[0]].x, cl->points[index[0]].y, cl->points[index[0]].z).null());
160 }
161
162 //Creates circle from 3 indexed points in a point cloud
163 void defineCircle(PointXYZRGB p1, PointXYZRGB p2, PointXYZRGB p3) {
164 dualCircle = Vec(p1.x, p1.y, p1.z).null()
165 ^ Vec(p2.x, p2.y, p2.z).null()
166 ^ Vec(p3.x, p3.y, p3.z).null();
167 circ = dualCircle.dual();
168 Pnt temp = circ * Inf(1) * circ;
169 for (int i = 0; i < 5; i++) {
170 circleCenter[i] = temp[i] / temp[3];
171 }
172 findNormal((Vec(p1.x, p1.y, p1.z).null() ^
173     ^ Vec(p2.x, p2.y, p2.z).null() ^
174     ^ Vec(p3.x, p3.y, p3.z).null() ^
175     ^ Inf(1)).dual());
176 calcRadius(Vec(p1.x, p1.y, p1.z).null());
177 }
178
179 private:
180 //Function that finds the cricle radius
181 void calcRadius(Pnt p) {
182 radius = sqrt(pow(p[0] - circleCenter[0], 2) +
183                 pow(p[1] - circleCenter[1], 2) + pow(p[2] - circleCenter[2], 2));
184 }
185
186 //Function that creates the plane that the cricle lies on
187 void circle::findNormal(Pnt pln) {
188 for (int i = 0; i < 5; i++) {
189 plane[i] = pln[i] / sqrt(pow(p[0], 2) + pow(p[1], 2) + pow(p[2], 2));
190 }
191 }
192
193 };
194 };
195
196 //Class that creates and handles cylinders with CGA
197 class cylinder{
198
199
200 //Since there is no representation of a cylinder in CGA a
201 //cylinder will be represente by a circle on a plane.
202 //The direction of the normal of the plane in the circle center
203 //will be the center-axis of the cylinder
204 public:
205 Pnt plan; //Store the IPNS representation of a plane
206 Pnt center; //Stores the center in a conformal vector
207 float radius = 11000; //Float for storing the radius
208 sphere s1, s2;
209
210
211 //Creates a cylinder from two points in conformal space on the center axis and a radius
212 void defineCylinder(sphere p1, sphere p2, float rad) {
213 radius = rad;
214 center = Vec(p1.x,p1.y,p1.z).null();
215 plan = Vec(p1.x-p2.x, p1.y-p2.y, p1.z-p2.z).null();
216 s1 = p1;
217 s2 = p2;
218 }

```

```
219  /*
220  //Creates a cylinder from 3 points in conformal space
221  void defineCylinder(Pnt p1, Pnt p2, Pnt p3) {
222  plane pln;
223  pln.defineDual(p1, p2, p3);
224  plan = pln.normDualPlane;
225  circle cir;
226  cir.defineCircle(p1, p2, p3);
227  center = cir.circleCenter;
228  radius = cir.radius;
229 }
230 */
231 //Creates the cylinder from 3 indexed points in a point cloud
232 void defineCylinder(PointCloud<PointXYZRGB>::Ptr cl, vector<int> index) {
233 plane pln;
234 pln.defineDual(cl, index);
235 plan = pln.normDualPlane;
236 circle cir;
237 cir.defineCircle(cl, index);
238 center = cir.circleCenter;
239 radius = cir.radius;
240 }
241
242
243
244
245 };
```
