Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Ajuste y clasificación de objetos hechos por el ser humano utilizando técnicas de sketching automático y restricciones geosemánticas

TESIS

Que presenta

Alberto Beltrán Herrera

Para obtener el Grado de

Doctor en Ciencias

en Computación

Directora de Tesis

Dra. Sonia Guadalupe Mendoza Chapa

Ciudad de México Febrero de 2019

Resumen

El reconocimiento de objetos hechos por el ser humano es una tarea que ha tomado gran importancia en los últimos años, gracias a las investigaciones en navegación automática, tanto en interiores como en exteriores, ya que estas requieren que vehículos y robots posean la capacidad de identificar e interactuar con el entorno urbano y casero, teniendo como prioridad el uso de algoritmos rápidos y robustos, y reconociendo una gran cantidad de objetos cotidianos, de forma casi instantánea. Los investigadores, que actualmente se encuentran trabajando en esta área, han dejado atrás los ejercicios académicos en los que era suficiente discernir entre una decena de objetos muy distinguibles, centrándose en el problema real de la navegación autónoma, que involucra clasificar cientos de objetos hechos por el ser humano. Esto implica extraer nuevas características de las imágenes que robots y vehículos usan para orientarse, permitiendo alcanzar nuevos hitos y mejores porcentajes en la clasificación. El presente trabajo utiliza sensores de profundidad junto con imágenes tradicionales (imágenes RGBD), aumentando la información del sistema. Este enfoque es usado por varios autores en la literatura, quienes lo combinan con técnicas tradicionales, como descriptores de imágenes, para buscar ligeras mejoras en la asertividad. Nuestro enfoque sigue un camino distinto en aras de mejorar la asertividad del proceso de clasificación, aprovechando las características del diseño industrial de los objetos hechos por el ser humano, estructurados a partir de primitivas tridimensionales. Usando la información de la capa de profundidad, es posible utilizar técnicas basadas en *sketching* para ajustar primitivas tridimensionales al objeto, de manera automática, y apegarse a los principios del diseño industrial, mediante restricciones geosemánticas, durante el proceso de ajuste, obteniendo así una reconstrucción certera de su estructura. De esta manera, en lugar de utilizar características de la imagen para el proceso de clasificación, se pueden usar las características de las primitivas tridimensionales que componen un objeto, mejorando la asertividad en la clasificación con una técnica que, además, resulta invariante ante transformaciones geométricas y factores, como sombras, luces y ruido en la imagen.

Palabras clave: objetos hechos por el ser humano, clasificación, *sketching*, restricciones geosemánticas, primitivas tridimensionales, ajuste de primitivas.

Abstract

Recognizing human made objects is one of the most important tasks in the last years because of its relevance in all the autonomous navigation research projects. Both indoor and outdoor navigations are required by robots and vehicles with the capacity to identify and interact with urban and in-house environments, using fast and strong algorithms as priority, and recognizing in real time a big amount of the daily life objects. Researchers that are currently working on this area have left behind the academic exercises where it was enough for them to pick out between just ten objects that were extremely different from each other. Now they are concentrating on the real problem of autonomous navigation, that involves the classification of hundreds of human made objects. This implies to get new characteristics from the images, that robots and vehicles use to navigate, allowing us to achieve new goals and better classification ranks. This research work uses depth sensors along with traditional images (RGBD images), increasing the system's information. This mindset has been used before by a lot of other researchers in this topic, who use it along with traditional techniques, such as image descriptors, to achieve little improvement on the assertiveness. Our approach follows a different way, searching to improve the assertivity in the classification process, and taking advantage of the industrial design of human made objects, structured by tridimensional primitives. Using the information provided by the depth layer, we can use a sketching-based technique to fit tridimensional primitives to the object, in an autonomous way and sticking to the bases of the industrial design by geosemantic constrains in the process, getting an approach to the object's structure. In this way, instead of using the characteristics of an image to classify it, we use the characteristics of the tridimensional primitives that form the object, improving the assertivity on the classification with a technique that is also invariant to geometric transformations, shadows, lights, and noise in the image.

Keywords: man-made objects, classification, sketching, geosemantic constraints, three-dimensional primitives, primitives fitting.

Agradecimientos

A CONACYT por el apoyo proporcionado para continuar con mis estudios y que me permitió desarrollarme durante esta etapa de mi vida.

Al CINVESTAV por abrirme sus puertas y procurar mi formación en todos los aspectos.

A mi mamá (mita) aunque ya no estas conmigo, te debo todo lo que soy, te dedico esta tesis y todos los años de esfuerzo que puse en ella, tu fuiste mi pilar, me enseñaste a no rendirme y a ser fuerte, siempre creíste en mí, no existen palabras para agradecer todo lo que hiciste por mí.

A mi papá, gracias por darme fuerza cuando más lo necesitaba, he llegado tan lejos por todo lo que me has enseñado, eres un gran ejemplo para mí, espero ser un hombre tan grande como tú.

A mi novia Carla, gracias por ser mi compañera de batalla, me has dado fuerza y alegría, has peleado a mi lado contra todo lo que nos ha hecho frente, esta tesis no hubiera sido posible sin ti.

A mis hermanas Gris y Mariana, quienes me brindaron todo su apoyo, son muy importantes en mi vida y no podría haber pedido mejores hermanas, siempre podrán contar conmigo.

A mis amigos, Dulce, Checo, Anita y Temoc quienes ya son parte de mi familia, me han ayudado más allá de lo que cualquiera hubiera hecho, siempre les agradeceré todo lo que han hecho por mí.

A mi directora, la Dra. Sonia, quien es una maravillosa persona y a quien considero una gran amiga, hizo posible este trabajo y me apoyo mucho más allá de lo que cualquier director hubiera hecho.

A Sofi, quien siempre hizo ameno y agradable mi paso por CINVESTAV, no solo es una maravillosa secretaria sino también un mujer increíble.

Índice general

1 Introducción **1**
- 1.1 Antecedentes . . . 1
- 1.2 Planteamiento del problema . . . 4
- 1.3 Objetivos . . . 5
  - 1.3.1 Objetivo general . . . 5
  - 1.3.2 Objetivos específicos . . . 5
- 1.4 Metodología . . . 6
- 1.5 Resultados . . . 9
- 1.6 Organización de la tesis . . . 10

2 Estado del arte **13**
- 2.1 El bocetado o *sketching* y su historia en la Computación . . . 14
- 2.2 Adquisición de *sketches* . . . 17
- 2.3 Clasificación de *sketches* . . . 18
- 2.4 Reconstrucción a partir de *sketches* . . . 20
  - 2.4.1 Reconstrucción por ajuste de primitivas geométricas . . . 22
- 2.5 Trabajos relacionados . . . 23
  - 2.5.1 Geosemantic Snapping for Sketch-Based Modeling . . . 23
  - 2.5.2 3-sweep: Extracting editable objects from a single photo . . . 25
  - 2.5.3 How do humans sketch objects? . . . 27
  - 2.5.4 3D Sketch Recognition Using the Microsoft Kinect . . . 29

3 Adquisición de imágenes **31**
- 3.1 Cámara de profundidad . . . 32
- 3.2 Inpainting . . . 34
- 3.3 Matriz de transformación . . . 37

4 Extracción automática de *sketches* **43**
- 4.1 Extracción del fondo . . . 44
- 4.2 Normalización del histograma de profundidad . . . 47
- 4.3 Extracción de contornos y esquinas . . . 50
- 4.4 Emparejamiento de contornos y esquinas . . . 54

5 Segmentación convexa **57**

5.1 Grafo de puntos de interés . . . 58
5.2 Symmetric Hull . . . 64
5.2.1 Contexto geométrico . . . 65
5.2.2 Algoritmo Symmetric-Hull . . . 68
5.3 Descomposición por aproximación convexa . . . 72
5.3.1 Segmentación convexa sin huecos . . . 72
5.3.2 Segmentación convexa con huecos . . . 76

6 Ajuste de primitivas tridimensionales 81
6.1 Algoritmo RANSAC para el ajuste de primitivas tridimensionales . . . 82
6.1.1 Parámetro de iteración k . . . 86
6.1.2 Criterio de paro b . . . 86
6.2 Langragiano aumentado . . . 87
6.3 Primitivas tridimensionales . . . 90
6.3.1 Esferas . . . 91
6.3.2 Conos y cilindros . . . 93
6.3.3 Prismas rectangulares . . . 97
6.4 Selección de primitivas tridimensionales . . . 101
6.5 Restricciones geosemánticas . . . 104
6.5.1 Descripción y aplicación de las relaciones geosemánticas . . . 105
6.5.2 Relaciones geosemánticas como restricciones en la función objetivo . . . 106

7 Clasificación de objetos hechos por el ser humano 109
7.1 Vector característico . . . 111
7.1.1 Análisis de cilindros en objetos hechos por el ser humano . . . 112
7.1.2 Análisis de prismas rectangulares en objetos hechos por el ser humano . . . 113
7.1.3 Análisis de esferas en objetos hechos por el ser humano . . . 115
7.1.4 Forma del vector característico . . . 116
7.2 Banco de datos . . . 116
7.2.1 Descripción del banco de datos . . . 117
7.2.2 Construcción del banco de datos . . . 119
7.3 Resultados de la clasificación . . . 122
7.3.1 Estudio de la metodología de validación . . . 122
7.3.2 Estimación de resultados . . . 123
7.3.3 Estudio comparativo . . . 124

8 Conclusiones y trabajo a futuro 129
8.1 Análisis de la complejidad . . . 131
8.2 Discusión de resultados . . . 132
8.3 Trabajo futuro . . . 136

Bibliografía 139

Capítulo 1

Introducción

1.1 Antecedentes

El reconocimiento de objetos hecho por el ser humano es un reto que ha acompañado a la historia de las áreas de Reconocimiento de Patrones y de Visión por Computadora. Desde los años setenta, se tienen documentados los esfuerzos de investigadores por reconocer estructuras edificadas por el ser humano, ya sea mediante vistas aéreas, satelitales o submarinas [Lendaris and Stanley, 1970] [Dudani et al., 1977], con fines de espionaje y planificación militar. Posteriormente, estos esfuerzos evolucionaron en sistemas de reconstrucción de espacios urbanos, mediante mapas de altitud y ajuste de edificaciones. Sin embargo, estos primeros trabajos se enfocaban únicamente en la separación entre artilugios o estructuras fabricadas por el hombre y aquellas formas naturales que podían encontrarse en el fondo de una imagen. En general, este enfoque permitía obtener buenas aproximaciones, pues sólo se pretendía separar las fotografías con probables estructuras humanas de una multitud de imágenes tomadas en un recorrido, a fin de que este conjunto más pequeño pudiera ser analizado más tarde por expertos humanos.

Esta rama del Reconocimiento de Patrones comienza a adquirir importancia en la siguiente década, cuando el área de Visión por Computadora empieza a jugar un papel crucial en las nuevas metas de la Robótica y de los Sistemas de Navegación Automática: ya no sólo buscaban bases militares con fotografías aéreas y satelitales, pues estas nuevas metas planteaban sistemas más robustos capaces de navegar en ambientes domésticos y urbanos. Moldeada bajo las premisas de la ciencia ficción, la nueva generación de investigadores buscaba que las computadoras pudieran moverse de forma autónoma en el mismo ambiente que los seres humanos, interactuando de la misma forma en que nosotros lo hacemos.

En los ochentas, con base en los estudios realizados en la década anterior, cobra fuerza el enfoque geométrico, el cual busca aproximar primitivas geométricas bidimensionales y tridimensionales en imágenes obtenidas por cámaras o arreglos de cámaras para

simular profundidad, en el caso de imágenes estéreo. Este enfoque permite representar un objeto, mediante las primitivas geométricas que lo conforman, lo que resulta adecuado para objetos con una geometría ingenieril, i.e., piezas modeladas por el hombre con base en el siguiente principio de diseño: una forma sigue una función, dando como resultado piezas con una estética basada en curvas, rectas y, en general, primitivas geométricas. De hecho, estos objetos son los que comúnmente se pueden encontrar en un ambiente urbano. El ajuste de primitivas (*fitting*) se lleva a cabo después de un procesamiento de imágenes simple, en el cual se obtiene el contorno y una serie de puntos para hacer el ajuste de primitivas geométricas. Dado el poder de cómputo existente en aquella época, sólo era posible hacer el ajuste de formas básicas, así que la mayoría de las propuestas sólo ajustaba cuboides, esferas o elipsoides, pero esta aproximación era suficiente para las primeras interacciones entre una computadora y el ambiente que la rodeaba.

El enfoque geométrico debe su auge, en gran medida, a la aparición de dos algoritmos importantes en el reconocimiento de formas, que pueden ser representados mediante una ecuación. El primer algoritmo es la transformada de Hough [Paul, 1962], que es un proceso iterativo que permite descubrir primitivas geométricas en imágenes digitales, a partir de los puntos de contorno. Se limitaba sólo a rectas, lo que mermaba su uso entre la comunidad científica, pero fue en 1981 cuando Dana Ballard popularizó la transformada de Hough entre la comunidad de Visión por Computadora [Ballard, 1981], la cual estaba basada en la primera generalización desarrollada por [Duda and Hart, 1972]. El segundo algoritmo importante de la época fue RANSAC [Fischler and Bolles, 1981]. Se trata de un método iterativo que permite calcular los parámetros de la ecuación o modelo de un objeto, tomando puntos aleatorios del área ocupada por la misma y aproximando la solución. Ambos algoritmos permitían ajustar primitivas geométricas en áreas limitadas por el contorno de un objeto. Su uso se hizo extensivo a sabiendas de que, al conocer las primitivas geométricas de un objeto, su clasificación y reconstrucción entregaban resultados sobresalientes, por lo que el enfoque geométrico dominaría durante más de una década.

El uso del enfoque geométrico se detendría a mediados de los noventa, como resultado de las condiciones específicas que eran impuestas por las propuestas que trabajaban con este enfoque. Estas limitaciones eran el resultado del problema de la invariancia [Morel and Yu, 2009], según el cual se afronta a los métodos de clasificación de objetos a las diferentes condiciones a las que fue expuesto el objeto al tomarse la imagen. Por ejemplo, las condiciones de iluminación, sombras, contraste, brillos, texturas, ángulo de visión, colores, entre muchos otros factores, pueden afectar a una imagen, provocando la incorrecta clasificación o reconstrucción de la misma. Estas condiciones exponen el lado débil de la metodología geométrica, pues esta depende de que la extracción del contorno sea correcta y con una cantidad mínima de ruido. Se puede pensar en un objeto común, tal como una caja envuelta en papel para regalo; por lo general, estos papeles tienen texturas complicadas y diferencias de colores que hacen de la extracción de bordes una labor poco realista con los métodos tradicionales, como el de Canny y los métodos de gradiente. De hecho, la cantidad de bordes detectados

sería inmensa y el ajuste de una sola caja sería un proceso sumamente complicado; además un inconveniente tan simple, como una sombra proyectada sobre un objeto, puede confundir fácilmente a cualquier detector de bordes y hacer que arroje, como resultado, la detección de dos o más objetos cuando sólo hay uno.

El problema de la invariancia fue objeto de interés de múltiples investigadores dedicados al reconocimiento y a la reconstrucción de objetos hechos por el ser humano. Por lo tanto, muchas propuestas existen en la literatura científica, tales como modelos fractales, modelos probabilísticos, conjuntos de nivel, técnicas de clasificación *clustering* y múltiples propuestas basadas en la detección de esquinas. No sería sino hasta finales de los noventa y principios del nuevo milenio que el enfoque más relevante aparecería: los descriptores de imágenes. La idea general de un descriptor de imágenes es representar a una imagen mediante ciertas características invariantes a las condiciones con las que dicha imagen fue tomada. Estas características son representadas por valores numéricos de las diferentes regiones de una imagen, lo que da como resultado un vector característico que pasa a ser la entrada de un algoritmo de clasificación. Si bien el campo de los descriptores de imágenes es sumamente amplio, son los descriptores HOG [Dalal and Triggs, 2005], SIFT [Lowe, 1999] y SURF [Bay et al., 2006] los más utilizados, cada uno invariante a diferentes condiciones, pero ninguno hasta el momento, invariante a todas. Los descriptores de imágenes permitieron un gran avance en el reconocimiento de objetos hechos por el hombre y, en algunos trabajos, también en la reconstrucción, ya que en conjunto con otras técnicas hicieron posibles sistemas robustos de navegación en ambientes urbanos y en interiores.

A partir de 2010 aparecería una serie de cámaras RGB-D (i.e., cámaras RGB con un sensor de profundidad incorporado) comerciales, de bajo costo y accesibles a todo público, como Microsoft Kinect. Estas facilitaron, en gran medida, la investigación en reconocimiento y reconstrucción de todo tipo de objetos, pues la información de profundidad, asociada a cada píxel, mejoraba las capacidades de reconstrucción y daba nuevos datos a los clasificadores. Las técnicas que se habían vuelto tradicionales se adaptaron rápidamente a estas nuevas capacidades en hardware. Si bien la nueva capa de profundidad generaba mejores resultados en la clasificación y reconstrucción de objetos hechos por el ser humano, estos rápidamente alcanzaban un límite asintótico del 75 % de asertividad, por lo que un nuevo enfoque era necesario.

En los últimos años, los investigadores han propuesto el uso de datos de profundidad, no sólo como datos adicionales que permiten mejorar los índices de asertividad, sino como una ayuda en el procesamiento de imágenes, obteniendo como resultado "no contornosçomo hasta la fecha se habían procesado. Más bien se trata de la obtención de bosquejos o mejor conocidos por su nombre en inglés *sketches*, i.e., representaciones lineales monocromáticas de un objeto, que sólo plasman las características más relevantes del objeto. Estas características son las mínimas necesarias para que un objeto pueda ser reconocido por el ser humano, omitiendo detalles como textura, material, luces, sombras y cualquier otro que no forme parte de la estructura esencial del objeto. Esto es posible gracias a que la capa de profundidad puede ser tratada

por separado y presenta propiedades distintas a las de la capa RGB. Mientras la primera sólo percibe distancias (con cierta cantidad de ruido), la segunda se centra en los colores. Las desventajas de la capa RGB, respecto a la invariancia, pueden ser compensadas por la capa de profundidad donde la textura, el color, el brillo, el contraste, las sombras y luces no tienen lugar, mientras que el ruido de los sensores de profundidad puede ser compensado por la capa RGB [Malik et al., 2016]. Esta nueva tendencia a utilizar *sketches* para el reconocimiento, en lugar de contornos simples, ha hecho resurgir el enfoque geométrico, que ya no ve mermada su asertividad por el problema de la invariancia. Como resultado, los objetos de geometría ingenieril pueden ser claramente vislumbrados, utilizando ajuste de primitivas.

1.2 Planteamiento del problema

La problemática actual del reconocimiento de objetos hechos por el hombre involucra el reconocimiento de cientos de objetos comunes, ya sea para ambientes en exteriores o en interiores. Nos topamos entonces con el hecho de que reconocer menos de una decena de objetos es, en muchas ocasiones, sólo un ejercicio académico. Por ejemplo, un sistema de navegación, de investigación o de domótica, se encontraría con la difícil tarea de tener que interactuar con cientos de objetos hechos por el hombre, sin poder darse sólo una interacción por defecto entre ellos. Así, es necesario que los sistemas de cómputo reconozcan su ambiente a detalle para poder tomar las decisiones de interacción correctas. Al tratarse un sistema básico para que sistemas más complejos puedan llevar a cabo sus funciones, el sistema debe ser ligero respecto al poder de cómputo necesario, pero lo suficientemente asertivo para reconocer a la mayoría de los objetos a su alrededor.

La geometría ingenieril de los objetos hechos por el ser humano ofrece grandes ventajas aprovechables por los procesos de reconocimiento y reconstrucción de objetos. Así que es tarea de esta tesis crear una metodología que permita unir estos dos procesos, dando como resultado una reconstrucción aproximada basada en las primitivas tridimensionales que conforman un objeto fabricado por el ser humano. Así mismo, esta tesis propone un método de clasificación basado en las características geométricas del objeto, que permitan su correcta categorización. La unión de los procesos de reconstrucción y reconocimiento debe ahorrar recursos computacionales y aportar información a la clasificación.

Si bien existen muchos métodos para resolver el reconocimiento de objetos con geometría ingenieril, la premisa debe ser una reconstrucción eficiente respecto al poder de cómputo, utilizando únicamente información con la máxima entropía. Por esta razón, se propone el uso de la metodología geométrica para aproximar un objeto a sus primitivas tridimensionales básicas. Esto es posible gracias a los avances en hardware que permiten extraer el *sketch* del objeto, el cual tiene sólo la información más relevante, en vez de los simples de la imagen, como se había hecho durante más de cuatro déca-

das. De esta manera, se dispone de la información del objeto con la máxima entropía y, al mismo tiempo, con el menor ruido.

A diferencia de otros trabajos similares en la literatura científica reciente, la reconstrucción y la clasificación deben estar ligadas, pues los resultados de la reconstrucción deben ayudar a la correcta clasificación. Todo el proceso debe ser completamente automático, una vez fijados los parámetros de todos los algoritmos involucrados, i.e., estos deben tomar las decisiones sobre qué primitiva geométrica debe colocarse en una sección del *sketch*, así como dividirlo en las secciones de mayor entropía para representar al objeto con un nivel de detalle aceptable, sin rayar en niveles tan específicos que saturen el poder de cómputo.

Para la extracción automática de *sketches*, sólo debe ser usado hardware comercial de fácil acceso para que este trabajo pueda ser reproducido por otros investigadores. Así, el uso del Microsoft Kinect® es altamente recomendado.

1.3 Objetivos

1.3.1 Objetivo general

Este trabajo tiene por objetivo el desarrollo de un sistema que abarque los procesos de reconstrucción y clasificación de objetos fabricados por el ser humano, integrándolos en un proceso que aproveche las características de ambos para aminorar el costo computacional. Este sistema debe ser robusto ante el problema de la invariancia y eficiente, de forma que pueda ser utilizado por sistemas que requieren obtener rápidamente información del medio tridimensional en el que se desenvuelven, con el fin de reconocer con exactitud los objetos a su alrededor y aproximar su forma para decidir la interacción que deben tener con ellos. El proceso propuesto debe ser llevado a cabo de forma automática, una vez ajustados los parámetros del modelo, lo cual es un factor diferencial con los actuales trabajos relacionados, cuya principal desventaja es la constante supervisión del usuario. El segundo factor de diferenciación es el uso de primitivas tridimensionales, las cuales se ajustarán a la forma del objeto, creando una aproximación aceptable, al mismo tiempo que sirven como atributos para la clasificación. Así mismo, con el fin de reducir el poder de cómputo necesario, se hace uso de *sketches* obtenidos mediante cámaras RGB-D, como una abstracción del objeto visualizado, de manera que los procesos de reconstrucción y clasificación utilicen únicamente la información del objeto con mayor entropía.

1.3.2 Objetivos específicos

1. Proponer un sistema de extracción de *sketches* de forma automática (sin intervención humana) a partir de un imagen RGB-D tomada mediante Microsoft Kinect®. Este sistema debe extraer el contorno del objeto, utilizando datos de

profundidad y RGB, con el propósito de evitar la mayoría de las complicaciones descritas por el problema de la invariancia.

2. Realizar e implementar una metodología de aproximación de objetos tridimensionales, mediante primitivas geométricas que se ajusten a los diferentes espacios de un *sketch* tridimensional.
3. Desarrollar un algoritmo de descomposición de formas convexas para automatizar la selección de espacios de ajuste de primitivas, con base en la propuesta de [Chen et al., 2013].
4. Automatizar el proceso de selección de primitivas tridimensionales mediante RANSAC u otra aproximación que permita seleccionar el mejor modelo para un espacio convexo en función del error presentado.
5. Implementar restricciones geosemánticas al ajuste de aproximaciones por primitivas tridimensionales, de forma que el modelo resultante tenga una geometría ingenieril.
6. Crear un banco de objetos fabricados por el ser humano, representados por las primitivas tridimensionales que los conforman. Los objetos seleccionados estarán basados en las etiquetas de los objetos más comúnmente utilizados que fueron propuestas por [Russell et al., 2008], de modo que se considere un banco de datos extenso, al incluir por lo menos cincuenta objetos distintos.
7. Evaluar los diferentes métodos de clasificación que la literatura ofrece para el problema de reconocimiento de *sketches*, con el fin de determinar el método con los mejores resultados, que se destaque por una relevancia estadística.
8. Proponer un método de clasificación adecuado a un banco de datos construido a partir de las primitivas tridimensionales que se ajustan a un objeto, sus características y relaciones.
9. Medir la asertividad de la clasificación propuesta en el reconocimiento de objetos hechos por el ser humano, así como el error en la aproximación por ajuste de primitivas tridimensionales.

1.4 Metodología

Siendo un sistema altamente complejo el que se desea proponer, es sumamente importante establecer el camino a seguir, ya que la investigación en las áreas de reconstrucción y clasificación de objetos hechos por el ser humano es muy basta. Existen muchas aproximaciones con resultados interesantes, que siguen diferentes metodologías. También hay muchas otras que son sólo derivaciones de los principales trabajos reportados y algunas otras que han demostrado los caminos que no deben seguirse. Para trazar nuestro camino, es necesario recalcar que el objetivo de nuestro trabajo es obtener resultados sobresalientes de clasificación y una reconstrucción aceptable basada en primitivas tridimensionales, así que ambos procesos deben ser lo suficientemente rápidos

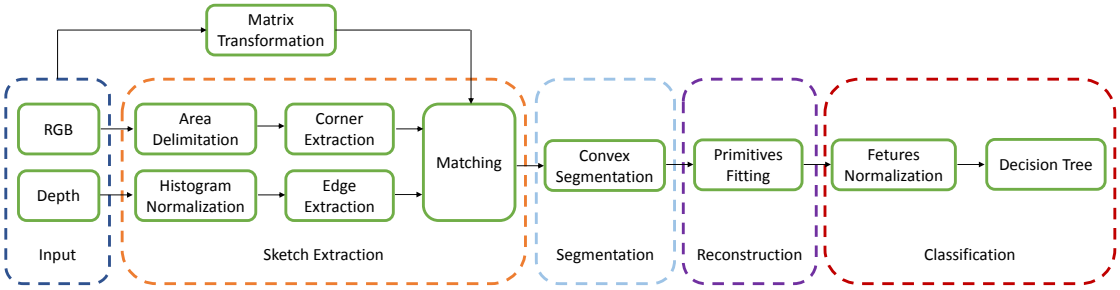


Figura 1.1: Etapas de la metodología del presente trabajo

como para permitir la navegación en un espacio tridimensional. Bajo esta premisa, se presenta la Figura 1.1, que actúa como mapa de navegación sintetizado de esta tesis.

La primera etapa es la adquisición de la imagen, la cual en nuestro caso particular se hará mediante Microsoft Kinect v2® por ser, al momento de comenzar esta tesis, la cámara RGBD comercial que ofrece la mejor definición, la mayor extensión comercial y una API robusta para la adquisición de imágenes. Afortunadamente, Microsoft Kinect® entrega automáticamente la imagen separada en dos capas, la RGB y la capa de profundidad. Sin embargo, los sensores no tienen la misma capacidad, no se encuentran en la misma posición, ni presentan la misma relación de aspecto, por lo que es necesario encontrar una transformación entre cámaras. Así, la Figura 1.1 hace referencia a una etapa externa de calibración de cámaras, que nos permitirá encontrar la matriz de transformación entre las imágenes obtenidas por la cámara RGB y las obtenidas por la cámara de profundidad (para factores de traslación, rotación, escala, sesgo, etc.). Como este proceso sólo debe hacerse una vez, se denota como un proceso externo, ya que la matriz de transformación prácticamente no variará a lo largo del tiempo.

La segunda etapa consiste en el procesamiento digital de las imágenes tanto en el espacio RGB, como en el espacio de profundidad. Si bien ambos procesos deben comenzar por separado, el resultado debe ser único, al obtener características de ambos. Para ello, se deben aplicar diferentes valores para el adecuado umbralado (o en su defecto la transformación adecuada en escala de grises), así como diferentes algoritmos para obtener los bordes o contornos. A este respecto, es necesario recordar que se tienen dos casos distintos. Por un lado, las imágenes RGB son susceptibles a los errores resultantes del problema de la invariancia, pero tienen bien definidos los contornos, por lo que se obtendrán bordes con líneas suaves. Por otro lado, las imágenes de profundidad no tienen problemas de invariancia, pero tienen una gran cantidad de ruido, así que los bordes no serán suaves. Es a partir de las ventajas de ambas imágenes que puede formarse el *sketch* del objeto, obteniendo sólo los bordes relevantes que lo formarán, sin tener en cuenta luces, texturas o sombras. El estado del arte sobre detectores de bordes es extenso, desde los trabajos de [Harris and Stephens, 1988] y [Canny, 1986] hasta los complicados bordes jerárquicos con características globales y locales de [Arbelaez et al., 2011], por lo que es necesario un análisis extenso sobre qué

método es lo suficientemente rápido y robusto, dadas las características del problema. La salida de esta etapa debe ser un único *sketch* tridimensional que represente al objeto.

Nuestras principales contribuciones comienzan a partir de la tercera etapa, señalada en la Figura 1.1 como la Etapa de Segmentación. Si bien es posible utilizar un método robusto para la segmentación de un objeto en sus partes convexas, como el propuesto por [Liu et al., 2010], este puede tomar una cantidad de recursos computacionales considerables. Además, la separación convexa es un problema *NP-Hard*, por lo que muchas de las técnicas se vuelven problemas de programación lineal o metaheurísticas. Dadas las limitaciones de nuestro problema (i.e., la geometría ingenieril correspondiente a los objetos fabricados por el ser humano), el problema no es tan complicado, ya que las regiones convexas son amplias y más fáciles de identificar. Por lo tanto, no debe aplicarse un algoritmo genérico, sino desarrollar uno propio que, basado en una técnica simple como las cubiertas convexas o la triangulación de Delaunay [Lien and Amato, 2006], pueda entregar resultados aceptables, siempre que haya una geometría ingenieril involucrada.

Una vez conocidas las zonas convexas del *sketch*, es necesario hacer el ajuste de las primitivas tridimensionales básicas que se definan (e.g., prismas, esferas, pirámides, cilindros y conos) en las zonas convexas obtenidas en la tercera etapa. El ajuste de múltiples modelos podría ser un cuello de botella importante, a tal grado que el algoritmo podría tardar varios minutos antes de entregar un resultado. Sin embargo, es aquí donde aparece la importancia del uso de *sketches*, pues gracias a que el *sketch* sólo tiene los puntos con máxima entropía del sistema (todos aquellos bordes relevantes) el ruido es mínimo, lo que permite hacer un ajuste de primitivas tridimensionales, usando pocos puntos de contorno. A diferencia de otros trabajos relacionados, no es necesario hacer crecer las regiones convexas desde el interior, pues la ubicación de los bordes relevantes es conocida. Así, es posible modificar el algoritmo de [Fischler and Bolles, 1981] para evaluar diferentes primitivas tridimensionales, utilizando pocos puntos y pocas iteraciones para descartar rápidamente las primitivas tridimensionales que no encajan. Los *sketches* presentan poco ruido, por lo que también es factible utilizar el método de Newton para aproximar las funciones objetivo de las primitivas tridimensionales, logrando un buen ajuste, pero debe tenerse en cuenta que se necesita una medida de error entre modelos. Es en este punto que, además de identificar las primitivas tridimensionales, estas deben ser ajustadas, respecto a las primitivas vecinas que conforman el modelo total del objeto. Para ello, será necesario implementar las restricciones geosemánticas que ayudarán a que el resultado de esta etapa sea un modelo completo del objeto conformado por primitivas tridimensionales, que se encuentran interconectadas entre sí, i.e., que presentan una geometría ingenieril.

Una vez que las primitivas tridimensionales que conforman el modelo son conocidas, se pueden tomar las características de estas (tales como el número de cada primitiva, su volumen, posición relativa, interacción con otras primitivas, circularidad, radio, alto,

ancho, largo, centroide y rotación en los tres ejes) como atributos para un proceso de clasificación. La lista de atributos que ofrece el ajuste de primitivas es bastante amplia, pero no todos los atributos son necesarios. Por lo tanto, en esta etapa se requiere hacer un análisis de atributos, además de que todos ellos deben ser normalizados para que las diferencias de rangos o posiciones no afecten el proceso de clasificación. De igual manera, será necesario determinar el mejor método de clasificación. Si bien las máquinas de soporte vectorial [Cortes and Vapnik, 1995] o las redes neuronales [Ha and Eck, 2017] parecen una opción viable, según la literatura científica, lo cierto es que si analizamos la lista de atributos de las primitivas tridimensionales, puede verse que varios de ellos son datos categóricos o caen en un rango limitado, por lo que vale la pena explorar alternativas, como los arboles de decisión. El resultado de esta etapa es un proceso de clasificación que debe ser rápido (por ello no se pueden seleccionar algoritmos de clasificación perezosos como K-means [MacQueen et al., 1967]) y muy exacto pese a tener que identificar una gran cantidad de objetos.

1.5 Resultados

En respuesta al planteamiento del objetivo general, como resultado de esta tesis se presenta un sistema que permite reconstruir y reconocer objetos hechos por el ser humano (con geometría ingenieril) con un índice de asertividad superior al reportado en la literatura científica, que se ejecuta de forma eficiente y que procesa sólo la información con la máxima entropía. Estas características permiten que pueda ser utilizado por sistemas de navegación y de visión por computadora en general, sin tener que gastar una gran cantidad de recursos computacionales en la clasificación y aproximación del modelado del objeto.

El banco de *sketches* formado por los atributos geométricos de las primitivas tridimensionales, que conforman al objeto que está siendo representando, es uno de los resultados planificados que queda abierto a la comunidad, no sólo para verificar los resultados presentados en esta tesis, sino que podría servir como punto de partida para desarrollar nuevos modelos de clasificación para *sketches* o como comparativa para los nuevos trabajos que surjan en el área.

En el desarrollo de la metodología, nos encontramos con varios problemas que no habían sido resueltos a plenitud en el estado del arte, como es el caso de la extracción automática de *sketches* a partir de una fotografía. En nuestro caso, este proceso se lleva a cabo utilizando una cámara RGBD, con resultados sobresalientes, por lo que queda, como aporte a la comunidad, un método que permite la extracción de *sketches*, el cual puede ser adaptado a cualquier hardware.

Otro de los problemas enfrentados es la segmentación de áreas convexas del *sketch*, proceso que es computacionalmente costoso por tratarse de un problema *NP-hard*. Dadas las limitaciones de los objetos que queremos reconocer (una geometría ingenieril), este problema puede ser resuelto de una manera más sencilla. La anterior

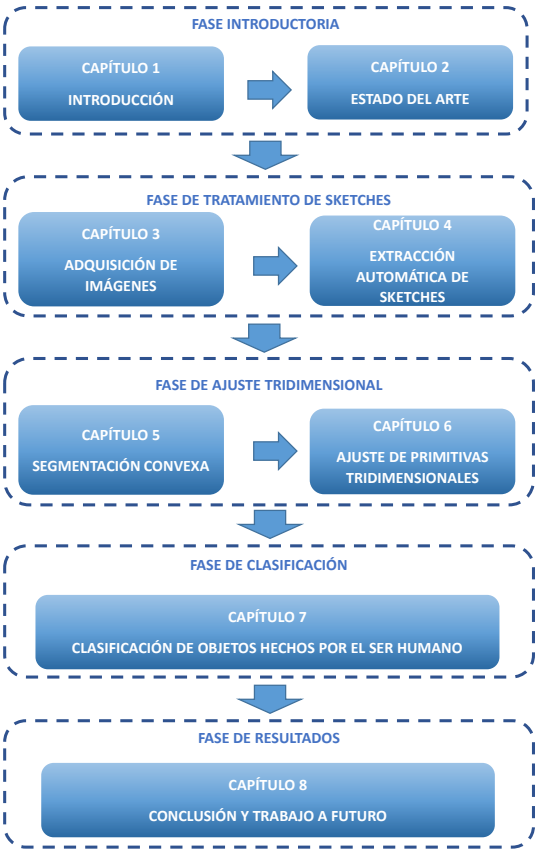


Figura 1.2: Organización de la tesis

afirmación desembocó en un algoritmo de segmentación convexa especializado en objetos con geometría ingenieril, que se basa en un algoritmo de cubiertas convexas, el cual también hemos desarrollado con este fin. Este último tiene la característica de lograr un tiempo superior a los conocidos hasta el momento en la literatura científica. Además, dicho algoritmo puede medir la diferencia de ángulos entre puntos continuos, siendo posible establecer umbrales para determinar el ruido en la cubierta convexa.

1.6 Organización de la tesis

Para una rápida revisión, el lector podrá encontrar un resumen de la organización del presente trabajo en la Figura 1.2.

En el Capítulo 2, se presenta un análisis del estado del arte, el cual está conformado por dos partes: marco teórico y trabajos relacionados. El marco teórico se centra en el desarrollo histórico del reconocimiento y reconstrucción de *sketches*, así como en un análisis de cuándo y por qué los trabajos existentes obtienen resultados que asintóticamente se estancan alrededor del 80 % de asertividad, lo cual da lugar a este trabajo.

A partir del Capítulo 3, se explica el trabajo desarrollado en la presente tesis, aunando en los aspectos matemáticos, científicos y tecnológicos que llevaron a este trabajo a obtener resultados competitivos. En específico, este capítulo habla acerca de cómo las imágenes son obtenidas desde las cámaras RGB y de profundidad, y cómo encontrar la matriz de transformación que permite pasar del espacio de profundidad al espacio RGB y viceversa.

El Capítulo 4 permite conocer uno de los principales diferenciadores con los trabajos relacionados hasta ahora analizados, pues da a conocer el proceso por el cual se lleva a cabo la creación automática de *sketches* a partir de fotografías RGB e imágenes de profundidad. El problema de invariancia no en un impedimento para realizar este proceso, gracias al aprovechamiento de las ventajas de ambos espacios y a la obtención de un *sketch* con coordenadas en tres dimensiones, el cual presenta una cantidad muy pequeña de ruido y cuyos trazos poseen la máxima entropía, al describir únicamente las características más relevantes del objeto.

En el Capítulo 5, se analiza la segmentación convexa, proceso que permite ajustar las primitivas tridimensionales (que tienen la propiedad de ser objetos convexos) en zonas específicas del *sketch*. Es importante destacar que el algoritmo utilizado para la descomposición convexa de las partes del *sketch* es completamente nuevo, siendo uno de los resultados destacables de esta tesis pues, aunque la descomposición convexa es un problema *NP-Hard*, la geometría ingenieril de los objetos hechos por el ser humano permite atacar este problema, mediante la descomposición por cubiertas convexas. Este enfoque no sólo facilita enormemente los cálculos necesarios. Además, dadas las características de los *sketches* generados en el Capítulo 4, fue necesario desarrollar un algoritmo de cubiertas convexas tolerante a ligeras variaciones en los grados de apertura y que tuviera la facilidad de medir numéricamente las diferentes aperturas. El resultado es una separación convexa intuitiva y poco demandante para la computadora.

Cada segmento convexo del *sketch* tridimensional debe ser ocupado por una primitiva tridimensional. Por esta razón, en el Capítulo 6, se presenta el proceso de ajuste de primitivas tridimensionales, el cual, mediante modificaciones a RANSAC, permite decidir sobre la primitiva tridimensional con mejor ajuste hacia el segmento convexo. Es en este capítulo donde se usan las restricciones geosemánticas para obtener una forma tridimensional, cuyas partes son congruentes entre sí, mostrando una geometría ingenieril.

Las primitivas tridimensionales ajustadas, según lo estipulado en el Capítulo 6, deben ser normalizadas para poder ser utilizadas como atributos de un clasificador. Esto se trata en el Capítulo 7, donde se describe el proceso de clasificación de un *sketch* dadas las primitivas tridimensionales que se ajustaron a él, así como los atributos geométricos utilizados, las pruebas y resultados obtenidos bajo diferentes esquemas de clasificación. En este capítulo también se describe la construcción del banco de objetos y se explica por qué algunos atributos son usados y otros no. Así mismo, se da una justificación del modelo de clasificación seleccionado y su porcentaje de

asertividad en bancos de datos que superan los 100 objetos.

El Capítulo 8 presenta las conclusiones y contribuciones a las que hemos llegado en el desarrollo de este trabajo. Se resumen las contribuciones alcanzados en la reconstrucción y clasificación de objetos, tanto en la percepción humana como con medidas de validación, mismas que son utilizadas por trabajos similares, lo que permite compararnos y establecer numéricamente las ventajas y desventajas. Este capítulo finaliza con ideas de trabajo a futuro, que dan pie a un campo fértil en el área de reconstrucción y reconocimiento de *sketches*, así como mejoras posibles al sistema y nuevas configuraciones que, en primera instancia, podrían mejorar el rendimiento o la eficiencia.

Capítulo 2

Estado del arte

Las tareas de reconstrucción y clasificación (o reconocimiento) de objetos tridimensionales a partir de imágenes son, por si mismas, campos fértiles de investigación. Aunque comúnmente se enfrentan a tareas relacionadas y ambas son necesarias para el desarrollo de sistemas de navegación automática, visión por computadora, sistemas geográficos etc., es muy común que ambas tareas sean tratadas por separado, ya sea para aumentar la precisión con la que cada una lleva a cabo su función o porque no comparten funciones en común. A pesar de que ambas tareas pueden partir de imágenes obtenidas por medio de algún sensor, dichas tareas no se comunican y generan interpretaciones separadas de la imagen para cumplir con su cometido. Esta separación de tareas, si bien puede mejorar los resultados de cada una, también genera una carga computacional alta, que se vuelve una problemática difícil de solucionar en sistemas cuya velocidad de respuesta es un factor crucial (como es el caso de los sistemas de navegación automática). Por esta razón, el presente trabajo propone una metodología que permita hacer una reconstrucción basada en primitivas tridimensionales básicas, mediante un proceso de *fitting*. Dicha metodología utiliza los parámetros de las primitivas obtenidas como valores de entrada para la clasificación, ahorrando importantes recursos computacionales, sin sacrificar de forma significativa la precisión de las tareas de reconocimiento y reconstrucción.

La afirmación con la que el párrafo anterior cierra, no sería posible si nos enfocáramos en cualquier tipo de objetos, pues su reconstrucción y clasificación tendría que converger en un clasificador capaz de reconocer cualquier objeto (tarea que suena titánica por sí misma); así mismo, se requieren reglas de reconstrucción tridimensional que puedan construir cualquier objeto sin importar su complejidad. Ambas tareas resultan retos monumentales, pues como puede verse en la literatura científica, los trabajos de reconstrucción y clasificación centran sus esfuerzos en problemas bien delimitados con objetos específicos y características bien definidas. Siguiendo una línea de pensamiento similar, este trabajo focaliza sus esfuerzos en trabajar únicamente con objetos fabricados por el ser humano, pues estos poseen características geométricas que les permiten ser descompuestos en primitivas geométricas básicas. De hecho, una gran

parte de los objetos de uso común pueden fraccionarse en una serie de cubos, esferas, cilindros, pirámides, prismas y conos, mientras objetos con formas naturales suelen ser mucho más difíciles de identificar a partir de imágenes, debido a las caprichosas formas que pueden tomar. Si bien se podrían usar técnicas, como modelos deformables, esto incrementa exponencialmente el cómputo necesario y anula la premisa de crear un procedimiento poco costoso computacionalmente para mezclar las tareas de clasificación y reconstrucción.

Dado que esta tesis se inscribe en las áreas de clasificación y reconstrucción de objetos hechos por el hombre, es necesario analizar cómo estas áreas han interactuado con el modelo de *sketches*, habiendo explicado antes la aparición, orígenes y desarrollo de este modelo en las Ciencias Computacionales. En el resto de este capítulo se explica la importancia del modelo de *sketches* y sus ventajas frente a otros, al responder las siguientes interrogantes:

- ¿Qué es un *sketch* en Procesamiento de Imágenes?
- ¿Cómo surgen los *sketches* y cuál es su importancia en las Ciencias Computacionales?
- ¿Cuáles han sido los caminos seguidos por los investigadores que han buscado mecanismos para mezclar la reconstrucción y clasificación de objetos específicos?
- ¿Por qué las representaciones por *sketches* se han vuelto tan populares?
- ¿Cuáles son las implicaciones de una clasificación de representaciones por *sketches*?
- ¿Cuáles son los alcances e implicaciones de una reconstrucción basada en *sketches*?

2.1 El bocetado o *sketching* y su historia en la Computación

Un *sketch* o boceto es una abstracción pictográfica de la realidad, la cual sólo conserva los suficientes rasgos característicos del objeto que representa, como para permitir su reconocimiento por los seres humanos [Johnson et al., 2009]. Esta definición es aceptada como genérica para el diseño, las artes, la Computación y otras ramas del conocimiento, aunque en el caso de la Computación es un término que fue adoptado al crear sistemas para el diseño que, a su vez, toman un término para una actividad ancestral del ser humano, como es el bocetaje o *sketching*.

El uso de formas de tipo *sketches* para representar la estructura básica de un objeto complejo, no es un tema reciente. En realidad, la humanidad ha utilizado *sketches* de objetos desde la época de las cavernas, donde los primeros representantes de nuestra especie pintaban la abstracción de los seres y condiciones de su medio. En fechas más recientes, las investigaciones de psicólogos, neurólogos y neurocientíficos aseguran

que haber comenzado con *sketches*, como abstracción de la realidad, no fue un evento fortuito. Por el contrario, resultó en un hecho inminente pues nuestro cerebro, al reconocer el *sketch* de un objeto, reacciona en forma muy similar al reconocer el mismo objeto del mundo real. Este hecho lleva a pensar a los científicos que nuestro cerebro se centra en reconocer formas abstractas, en vez de elementos complejos, como tradicionalmente se hace en Reconocimiento de Patrones y Visión por Computadora [Walther et al., 2011].

Sin las actuales investigaciones que probaban la importancia del *sketching*, como parte del entendimiento humano sobre el medio que lo rodea, esta técnica pasó a ser relegada como una previsualización de obras más complejas, sobre todo en las artes, donde se usaba (y sigue usándose) como una herramienta en el proceso creativo: un borrador que puede ser tachado, modificado y perfeccionado. Al ser clasificado como herramienta de las artes, no fue extraño que se comenzaran a desarrollar asistentes digitales que facilitaran la tarea de la creación de *sketches*, saliendo al mercado un gran número de herramientas de dibujo. Sin embargo, el poder del *sketching* no recae en simples herramientas de dibujo, sino en cómo estos pueden usarse como modelos de interacción, como fue el caso del afamado Sketch-pad [Sutherland, 1964], que por primera vez permitía una comunicación con la computadora mediante el uso de *sketches*, misma idea que más tarde sería refinada por el proyecto GRAIL [Ellis et al., 1969] y modificada para trabajar en 3D por [Johnson, 1963].

La estandarización de los periféricos de entrada y el limitado poder de cómputo para manejar sólo dibujos de dimensiones limitadas, frenaron momentáneamente la investigación, que sería retomada gracias a la llegada de las interfaces gráficas de usuario gráficas. En 1992 se daba a conocer el primer representante de la nueva generación de trabajos en el área, el sistema Viking, que permitía tanto la entrada de información en forma de *sketch*, como su reconstrucción [Pugh, 1992]. Poco después, en 1996, se presentaba en el congreso SIGGRAPH uno de los sistemas más citados de la literatura científica: SKETCH [Zeleznik et al., 1996], una interfaz que recibía comandos de entrada en forma de estructuras poliédricas. Así mismo, otras áreas como el Reconocimiento de Patrones encontraron oportunidad al tratar los *sketches* y clasificarlos, problema que hasta el momento sigue abierto. Por ejemplo, en el trabajo de [Shpitalni and Lipson, 1997], se usan las esquinas normalizadas de un *sketch* para ajustar secciones cónicas, mediante mínimos cuadrados, clasificando así estas secciones.

El incremento del poder de cómputo, los dispositivos *touchscreen* y las mejoras en las cámaras digitales, dieron pie a mayor investigación en el área, llegando a reportarse cientos de métodos de clasificación, reconstrucción y modelado basados en *sketches* [Olsen et al., 2009] ¿A qué se debe este apogeo por usar abstracciones de las imágenes, en lugar de las imágenes reales?

Mientras una imagen fotográfica tiene una gran cantidad de información que puede ser analizada, los *sketches* se caracterizan por contener únicamente la información más importante que permite a un sistema (humano o computacional) reconocer y traba-

jar con el objeto, como una abstracción de la realidad. Los *sketches* generalmente se forman por trazos monocromáticos fácilmente separables del fondo (propiedad que es aprovechada por muchos investigadores para realizar clasificación a partir del número y tipo de trazado [Plumed et al., 2013]). Esto permite hacer diferentes tipos de representaciones de un trazado, ya sea como imagen por píxeles, imagen vectorial, vector de puntos, líneas de trazado o representaciones poliédricas entre muchas otras. Estas representaciones contribuyen a expresar y resolver mejor un problema, siendo preferidas por la comunidad las representaciones vectoriales, por trazados y poliédricas, cuando se tiene acceso al muestreo *online* (como puede ser el trazado con una tableta digitalizadora o una pantalla táctil). Cuando únicamente se tiene acceso al trazado *offline*, la única opción disponible suele ser trabajar con imágenes tradicionales [Tumen et al., 2010].

Al tratarse de imágenes binarias con la información más relevante del objeto y sin ruido, estas pueden procesarse y almacenarse con muchas menos operaciones que las necesarias al tratar con imágenes fotográficas. Esto ayuda a reducir el costo computacional, pues los *sketches* pueden ser procesados y almacenados como matrices dispersas o inclusive *hash-bits* [Tseng et al., 2012].

Por otro lado, se sabe que las representaciones por *sketching* son inexactas, pues una abstracción de la realidad depende mucho de quien la haya realizado, el contexto socio-cultural, el estado anímico, condiciones anatómicas etc. [Arcavi, 2003], [Ibrahim and Rahimian, 2010]. Esto complica el trabajo de clasificación, pues la representación de un mismo objeto puede variar simplemente por la posición geográfica del dibujante, lo que representa un gran reto en el área. Esta misma idea lleva a otra problemática en el campo de investigación: el trabajo de reconstrucción, que dependerá en gran medida del propósito para el cual se requiere. Sistemas en los que se debe ser sumamente exacto con las reconstrucciones (como pueden ser los sistemas para arquitectura, medicina, videojuegos y *rendering*), buscan reconstrucciones basadas en vídeo o múltiples fotografías de alta calidad, estando dispuestos a pagar el poder de cómputo necesario para obtener escenarios bellamente reconstruidos. Por el otro lado, sistemas en los que el tiempo es una prioridad (como los sistemas de navegación, búsqueda de imágenes, o control automático) prefieren el uso de *sketches*, pues pueden obtener aproximaciones de buena calidad (sin llegar a la exactitud) que para sus propósitos tengan un valor representativo.

Como se ha ejemplificado hasta el momento, los *sketches* se afrontan a retos distintos a los que se enfrentan las imágenes convencionales. Si bien contienen ruido, la binarización, la dimensión, el almacenaje y el poder de cómputo necesario no representan problemas considerables, cuando se trabaja con *sketches* (tareas que sí lo representan al tratar con imágenes fotográficas). Sin embargo, si lo harán otras variables como la falta de generalidad en la descripción pictográfica abstracta, debido a las condiciones del dibujante como anteriormente se ha explicado. Así mismo el contenido de la información juega un doble papel, pues si bien durante el procesamiento ayuda a reducir el número de operaciones, también es posible que la información en un *sketch* no sea

la necesaria para la tarea en cuestión, por lo que el diseñador del sistema debe tener mucho cuidado al momento de trabajar con imágenes fotográficas o con *sketches*.

Este trabajo de tesis se encuentra dirigido a la creación de una metodología que permita la rápida reconstrucción y clasificación de objetos hechos por el ser humano, centrándonos en mejorar los resultados de la clasificación, mientras se entrega un esbozo de la reconstrucción basado en primitivas tridimensionales. Esto se debe a que los sistemas para los cuales se encuentra pensado (principalmente sistemas de interacción con el entorno) requieren tener una gran asertividad, cuando necesiten reconocer un objeto, mientras que las reconstrucciones les dan indicaciones de cómo interactuar con los objetos. Por esta razón, no es necesaria la exactitud del modelo, pues es suficiente con la estructura y forma.

2.2 Adquisición de *sketches*

Desde la aparición de SKETCH-PAD [Sutherland, 1964] en la década de los sesenta, se ha asociado a los dibujos de tipos *sketches*, con hardware específico. Pero no fue sino hasta la aparición de las interfaces gráficas de usuario, en la década de los ochentas, cuando se pudo comenzar a pensar en crear *sketches* en una computadora personal con hardware convencional (ratón de computadora). Desde entonces fue basta la investigación en técnicas y aparatos que permitían la adquisición de *sketches*. Particularmente, es interesante ver la división de estos en técnicas *online* y *offline*. Mientras que las técnicas *offline* realizan una adquisición por medios clásicos e incluso pueden ser hechos por los mismos medios que las técnicas *online* (cuando estos almacenan las imágenes de los *sketches*), su contraparte requiere generalmente un hardware especifico o un firmware que guarde las posiciones del trazo.

En el caso de las técnicas *online* es común el uso de lápices especiales, dispositivos de grabación en tiempo real, tabletas digitalizadoras, pantallas táctiles o programas de rastreo usando visión por computadora. Para los fines de las técnicas *online*, estos dispositivos juegan un papel sumamente importante, porque en estos métodos la entrada a la clasificación o reconstrucción, no es el *sketch* en sí, sino la forma en la que el *sketch* ha sido dibujado, matemáticamente lo podemos ver como que la entrada a un sistema *online* es un vector $v = p_1, p_2, \ldots, p_n$ donde cada p se compone de coordenadas (x, y) (recordando que hablamos únicamente de *sketches* en 2D). La conformación del vector puede ser muy diversa, mientras muchos trabajos toman la aproximación de utilizar los cambios de dirección [Beltrán and Mendoza, 2011], los cambios de trazo o el tipo de trazo, o transformar lo adquirido a un espacio conveniente [Bae et al., 2008]. Los modelos *online* terminan siendo idóneos cuando se requiere un procesamiento instantáneo del *sketch*, siempre y cuando el usuario cuente con el hardware y software necesario para ello, por lo que suelen ser la técnica preferida por los programas de dibujo y de reconocimiento por hardware.

Muchos de los problemas del mundo real, no permiten la adquisición de los *sketches*

de forma *online*, ya que no se cuenta con el equipo o software necesario, de hecho, tradicionalmente los bancos de datos de gran escala están formados por las imágenes recolectadas de cientos o miles de usuarios, sin que información adicional sea proporcionada, como el número de trazos o como estos fueron hechos. Este es el reto al que se enfrentan las técnicas *offline*, es decir, el tratamiento de *sketches* que fueron tomados con anterioridad y que generalmente vienen en forma de imágenes, lo que como resultada obvio resulta en un pre-procesamiento para separar sus partes o en el uso de descriptores de imágenes que permitan conocer los puntos de interés de la estructura [Eitz et al., 2012]. Es común que dado su manera de adquisición (desde el dibujo directo en la computadora hasta el escaneo o fotografías del *sketch*, vengan en formatos distintos, por lo que los autores generalmente deben de estandarizar los bancos de datos y después normalizar las imágenes para que los resultados, ya sea en clasificación o en reconstrucción, tengan resultados aceptables.

En los últimos años ha sucedido un fenómeno sumamente curioso gracias a la aparición de los sensores de profundidad (Microsoft Kinect, Leap Motion, Structure Sensor etc.), pues se ha adoptado estos como una nueva fuente de *sketches*, dibujando en el espacio 3D un *sketch* en 2D de forma muy natural, utilizando un lápiz especial o simplemente usando las manos, lo que resulta tan natural como pintar en una hoja de papel o en la pared, aunque estos gestos suelen hacerse en el aire [Taele and Hammond, 2014]. Algunos otros trabajos van más allá y toman el *sketch* en tres dimensiones formando un SBR de tres dimensiones que puede ser procesado utilizando descriptores tridimensionales para con la información extra, obtener mejores resultados en la reconstrucción [Li et al., 2015a] [Bulgerin, 2014].

2.3 Clasificación de *sketches*

En la Subsección 2.1 ya se hablaba de los primeros trabajos de clasificación de *sketches* en los 90, donde principalmente se hacía uso de las propiedades de los trazos o de las regiones que estos describían, para obtener valores numéricos que pudieran aplicarse a los clasificadores existentes, la gran mayoría de estas técnicas se conocen como *online*, pues están basados en separación de los trazos en el mismo momento en que estos son dibujados, así es posible hacer una segmentación a priori, generalmente estas técnicas son usadas cuando se conoce el hardware en el que serán implementadas. En aquellos momentos el poder de computo limitaba la contraparte de las técnicas *online*: las técnicas *offline*, que utilizan no los trazos al momento de realizarlos, sino las imágenes que ya han sido terminadas por algún usuario y generalmente en formato de imagen. Los números de trabajos reportados en clasificación de *sketches offline* se intensificarían gracias a los descriptores de imágenes, en especial de SIFT [Lowe, 1999] y HOG [Dalal and Triggs, 2005], que darían paso a un sinfín de trabajos sobre la clasificación de *sketches*, al permitir seccionar, normalizar y obtener valores numéricos de las distintas regiones de interés. El método general consiste en tomar una imagen y partirla en n sub-imágenes, generalmente se recomienda que n sea una potencia

de 2 con el fin de mejorar el procesamiento; a cada región se le aplica un descriptor de imágenes (el procedimiento es el mismo sin importar cual descriptor se aplique), con el valor (o valores según el descriptor usado) obtenido en cada región, se forma el vector característico, el cual generalmente tiene una dimensión elevada, pues para que la información del vector característico sea relevante, es necesario dividir lo suficiente para obtener una cantidad de información relevante, pero no tanto que las secciones no sean relevantes, por lo que es mérito del investigador proponer el correcto seccionado, sería común encontrar trabajos en la literatura con seccionados entre 128 y 1024, por lo que también suele ser común aplicar técnicas de selección de atributos como PCA y sus variaciones [Eitz et al., 2012].

El uso de los descriptores facilito la clasificación por técnicas como las máquinas de soporte vectorial (*Support Vector Machine* o SVM), redes neuronales y los K-vecinos más cercanos [Hu and Collomosse, 2013] [Bhatt et al., 2010], estos algoritmos resultaron fácilmente adaptables al campo de investigación de la clasificación de *sketches*, pues ocupaban que las imágenes estuvieran por lo menos en escala de grises (un *sketch* generalmente esta en monocromo) y las variaciones en la orientación y saturación de una sección ayudan a obtener una mejor descripción de la región analizada, variables que son bien definidas en la mayoría de las regiones de un *sketch*.

En los últimos años, nuevas investigaciones han llevado a desarrollar el área del reconocimiento de formas tipo *sketch*, lo que permitiría reconocer abstracciones del mundo real con gran exactitud, antes de intentar reconocer escenas reales usando un excedido número de recursos computacionales; el ejemplo más reciente y popular es conocido como "sketch-rnn", una red neuronal recurrente que ha sido entrenada por millones de bocetos de seres humanos a través de la página de internet: quickdraw.withgoogle.com, cuyo fin es aprender a reconocer toda una multitud de bocetos de objetos de la vida diaria [Ha and Eck, 2017]; este trabajo se diferencia del camino tradicional en el reconocimiento de *sketch* pues retoma la información de los trazos, al detectar como es que los seres humanos dibujan un cierto objeto, en lugar de hacerlo mediante las técnicas contemporáneas como es el uso de descriptores, técnica usada por innumerables trabajos actuales [Li et al., 2013] [Li et al., 2015b] [Schneider and Tuytelaars, 2014]. Si bien todos los trabajos recientes en clasificación de *sketches* se diferencian en la forma en la que se construyen los vectores característicos, así como en los algoritmos de clasificación utilizados, todos ellos tienen en común que intentan clasificar un gran número de diferentes objetos; antes de 2010 la mayoría de los trabajos se evocaban a distinguir solo entre un limitado número de *sketches*, 10 o menos, trabajos como el de [Paulson and Hammond, 2008] pueden obtener resultados de hasta 99 % de asertividad al clasificar menos de 9 objetos, sin embargo, estos son solo ejercicios académicos, pues los sistemas pensados para el mundo real tienen el actual reto de clasificar una cantidad de objetos mucho mayor, en el orden de las decenas o cientos de objetos [Eitz et al., 2012] [Schneider and Tuytelaars, 2014], lo cual disminuye drásticamente su asertividad, yendo desde el 60 % hasta el 75 % en los mejores casos, siendo estos indicadores lo que fomentan la actual investigación en el área. Una gran parte de la comunidad continua utilizando los descriptores de imágenes como principal método

para formar el vector característico de un *sketch*, sin embargo, es basta la literatura que ha demostrado que la caracterización por descriptores de imagen tiene un cierto límite [Eitz et al., 2011] [Cao et al., 2013], obteniendo resultados casi siempre por debajo del 70 % al tratar con más de 100 objetos a ser reconocidos. La explicación formal del problema de la asertividad frente a un gran número de clases (objetos a reconocer), se debe a que aunque dos *sketches* sean distintos, comparten trazados en común, lo que crea resultados numéricos muy similares al hacer el seccionado, lo que quiere decir que mientras mantengamos pocas clases de *sketches* a ser reconocidos y estos sean diferentes entre ellos, las técnicas por descriptores de imágenes funcionaran correctamente, pero a medida que el número crezca comenzara a haber más secciones con valores numéricos similares lo que provocará la incorrecta clasificación de varios *sketches*.

Al momento de escribir esta tesis, la clasificación de bancos de datos con una gran cantidad de *sketches* clasificados en más de 100 categorías, continua siendo un problema abierto.

2.4 Reconstrucción a partir de *sketches*

Desde 1960 el problema de la reconstrucción a partir de trazados simples o de *sketches*, ha encontrado en la computación, una herramienta que permite automatizar el proceso, siendo un problema conocido como reconstrucción geométrica, reconstrucción a partir de líneas de trazado o simplemente reconstrucción [Company et al., 2005]. Este problema se refiere al conjunto de técnicas capaces de tomar un *sketch* en 2D y crear un modelo 3D que se ajuste lo mejor posible a lo que el usuario ha querido representar con el *sketch*. Existe una gran variedad de técnicas para resolver la reconstrucción, de igual manera existen muchas clasificaciones como la de [Company et al., 2005] que clasifica los métodos de reconstrucción entre aquellos que usan un solo *sketch* con una sola vista del objeto 3D, contra aquellos que utilizan múltiples *sketches* con distintas vistas de un objeto en 3D. Una clasificación que ha sido más extendida en la literatura separa la reconstrucción en sistemas evocativos y sistemas constructivos [Olsen et al., 2009], la diferencia radica en que los primeros son conocidos por primero reconocer (o clasificar) el *sketch* y después ajustarlo a una o varias plantillas (*templates*), mientras que el segundo no hace ningún paso de clasificación y trata directamente de reconstruir el objeto en 3D; de igual manera existen algunos trabajos sobre sistemas híbridos en los que las plantillas se deforman para ajustarse al *sketch* o en los que los sistemas constructivos acotan su dominio a un selecto grupo de geometrías reconstructivas [Yang et al., 2005].

Los métodos evocativos dependen en gran medida de qué tan bien se haga la clasificación y de cuantos y qué tan buenas son sus plantillas, lo cual tiene implicaciones muy importantes cuando se identifican una gran cantidad de *sketches*, pues requerirá tener plantillas de todas las vistas posibles de un objeto 3D, además, recordando lo discutido en la Subsección 2.3, cuando tenemos una cantidad limitada de *sketches* a

reconocer tendremos resultados sobresalientes (99 % de exactitud) mientras que este porcentaje ira disminuyendo conforme aumentemos el número de *sketches*, llegando a porcentajes del 60 % o menos.

Tradicionalmente, los métodos evocativos se dividen a su vez en sistemas icónicos y sistemas de recuperación de plantillas. Los primeros se distinguen por identificar los trazos o un grupo de ellos, los cuales serán identificados como parte de una estructura geométrica en tres dimensiones, enfocándose únicamente en primitivas tridimensionales [Zeleznik et al., 2007] [Jorge et al., 2003]. Estos sistemas son en su gran mayoría *online* y asumen que las entradas son planas, simétricas y ortogonales (la última condición suele ser algo difícil cuando es un usuario con un trazo libre quien debe cumplir estas condiciones).

Los sistemas de recuperación de plantillas son más complejos que los sistemas icónicos pues en lugar de tomar primitivas tridimensionales, tratan de ajustar directamente plantillas de objetos complejos, lo que permite que las bases de objetos reconocibles sean fácilmente actualizables [Joaquim and Jorge, 2004]. La debilidad de esta aproximación es el poder de computo necesario, pues aunque puede entregar resultados interesantes, la comparación de una vista 2D con un modelo 3D requiere de varias comparativas de diferentes puntos de vista del modelo 3D almacenado; siendo clásicamente resuelto al obtener vistas 2D desde el objeto 3D o mediante el uso de descriptores invariantes a traslación, rotación y escala de un menor número de vistas. Típicamente estas aproximaciones se usan cuando el tiempo no es factor crucial y cuando se conocen todas las posibles respuestas del sistema de reconstrucción, su mayor uso recae en la reconstrucción de escenarios [Shin and Igarashi, 2007].

Por el otro lado tenemos los sistemas constructivos, que son mucho más complicados que los sistemas evocativos, pues en los últimos, el uso de plantillas ayuda a reconstruir un sistema a partir del conocimiento previo que se tiene de sus salidas, mientras los sistemas constructivos deberán proponer la geometría del objeto 3D a partir de reglas de diseño. Siguiendo la clasificación de [Company et al., 2005], los sistemas constructivos se pueden clasificar en tres categorías:

- Sistemas mecánicos o ingenieriles. Generalmente pensados para la reconstrucción de objetos hechos por los seres humanos, pues aplican procesos de ajuste (*fitting*) y optimización numérica para crecer planos y curvas, generalmente restringidos por condiciones de ortogonalidad y simetría. Este tipo de sistemas funciona bastante bien pues acota su campo de acción gracias a las condiciones geosemánticas del objeto deseado, aunque suele tener problemas con los segmentos curvos.
- Sistemas suaves o de reconstrucción libre. Esta categoría suele abarcar gran cantidad de trabajos que reconstruyen formas naturales para obtener interpretaciones 3D de un *sketch* 2D, que sean más agradables ante la vista y el cerebro. Dentro de esta categoría caen las aproximaciones basadas en esqueletos, triangulación de Delaunay, *blobs*, reconstrucciones topológicas etc., muchas veces ayudadas por *splines* u otros métodos de suavizado.

- Sistemas de múltiples vistas. Como su nombre lo indica se tratan de sistemas que como entrada necesitan de múltiples *sketches*, para aplicar técnicas similares a los de sistemas estereoscópicos que nos permitan parear diferentes trazos, líneas o puntos; formando una red de trazados que puede resolverse por un proceso de optimización numérica para encontrar su representación en el espacio 3D.

2.4.1 Reconstrucción por ajuste de primitivas geométricas

Clasificados dentro de los sistemas mecánicos o ingenieriles, la reconstrucción por ajuste de primitivas geométricas, había sido una herramienta ampliamente utilizada antes de la llegada de los descriptores de imágenes, pues hacer el ajuste o *fitting* de primitivas geométricas, implicaba conocer las primitivas tridimensionales que conformaban el objeto y su ubicación dentro de este, es decir, dejaban muchas de las decisiones al usuario [Eggli et al., 1997] o permitían que el usuario solo dibujara una primitiva tridimensional a la vez [Bloomenthal et al., 1998], al hacer una adquisición *online* de los *sketches*, no se preocupaban por la separación de las figuras, lo que facilitaba en gran medida el procesamiento.

El verdadero problema venia cuando se tenían que resolver problemas *offline*, pues estas técnicas dependían de que ya hubiera una separación o en el caso de versiones más automatizadas como la de [Chen et al., 2008], dependían de la calidad del trazo y la ortonormalidad de sus bases para garantizar resultados aceptables. Los *sketches* adquiridos como *offline*, encontraron una respuesta en los descriptores de imágenes, no solo en los tradicionales HOG y SIFT (aunque si son usados por una gran cantidad de trabajos), sino que muchos de los descriptores que habían sido utilizados para la reconstrucción de imágenes 3D a partir de imágenes 2D fueron aplicados a las técnicas de *sketching*, entre muchos otros los más aplicados han sido:

- *Fourier Spectra Descriptor from Silhouette, Contour, and Edge Images (EFSD)* [Aono and Iwabuchi, 2012]
- *Sketch-Based 3D Model Retrieval Based on 2D-3D Alignment and Shape Context Matching (SBR-2D-3D)* [Li and Johan, 2013]
- *Sketch-Based 3D Model Retrieval Based on View Clustering and Shape Context Matching* [Belongie et al., 2002]
- *Fourier Descriptors on 3D Models Silhouettes (FDC)* [Zhang et al., 2002]

Pese a tantos diferentes descriptores utilizados en la reconstrucción, aun se tiene el problema de que los métodos son lentos y no siempre presentaban resultados adecuados, ya que cuando se requiere un resultado rápido generalmente se acortan los métodos antes de obtener el mejor resultado posible. Como una respuesta a esta problemática, en 2013 [Shtof et al., 2013] propone retomar el uso de las técnicas constructivas mecánicas basados en primitivas tridimensionales básicas (esferas, prismas, conos, cilindros etc.) como acercamiento a la estructura original de un problema, siendo una alternativa realmente útil cuando buscamos la reconstrucción de piezas creadas

por el ser humano según lo demuestra en su trabajo. Este trabajo sería el detonador de varias obras que comenzarían a utilizar los modelos constructivo mecánicos como una aproximación acertada y rápida a los modelos abstraídos por los *sketches*.

Estas técnicas consisten en general en tomar un conjunto de primitivas tridimensionales y ajustarlas a las partes de un *sketch*, dependiendo del esquema se puede usar optimización mono-objetivo si lo que se quiere minimizar es solo el tamaño de la primitiva 3D, pero generalmente es más usual usar optimización multi-objetivo pues debemos ajustar al *sketch* diferentes variables del modelo de la primitiva tridimensional (como pueden ser el ancho, el alto, la profundidad, la rotación, traslación y escalamiento, así como diferentes ángulos internos), en el caso del trabajo de [Shtof et al., 2013], recae en el usuario ciertas decisiones, como la elección de la primitiva tridimensional que será la usada y la rotación que tendrá, por lo que el problema se acota a únicamente la dimensionalidad de la primitiva, lo cual se resuelve fácilmente al expandirla desde un punto cualquiera dentro de la región hacia los limites, esto midiendo el error cuadrático medio entre la geometría de la primitiva tridimensional y los bordes de la región del *sketch* que se está estudiando. Nuevos trabajos han sido desarrollados desde entonces proponiendo técnicas más eficientes para determinar las variables como es el uso de RANSAC, [Zhang and Chen, 2014] [Kratt et al., 2014], donde gracias al nivel de abstracción de los *sketches* y a que estos pueden tener una representación paramétrica, las iteraciones del algoritmo son realmente pocas comparadas con las necesarias por otros métodos.

2.5 Trabajos relacionados

2.5.1 Geosemantic Snapping for Sketch-Based Modeling

Basado en los trabajos de [Lau et al., 2010] y [Gingold et al., 2009] para la reconstrucción con base en primitivas tridimensionales, el trabajo de [Shtof et al., 2013] permite la reconstrucción de *sketches* a partir de primitivas tridimensionales básicas, específicamente este trabajo se centra en esferas, conos, cilindros y prismas. Su funcionamiento es simple, se carga el *sketch* en un área de trabajo donde el usuario tiene acceso a las diferentes primitivas tridimensionales (Figura 2.1), las cuales el usuario simplemente arrastra hacia la zona del *sketch* donde quiere que se ajuste la forma geométrica, repitiendo este proceso con todas las primitivas tridimensionales necesarias para modelar el *sketch* en pantalla.

Este trabajo se destaca por agregar dos aspectos importantes a sus antecesores, primero, es un sistema mixto, entre SBR y SBM, ya que toma un *sketch* y no hace la reconstrucción automática, sino que delega al usuario aquellas tareas en la que la computadora no es buena, es decir la clasificación y decisión de donde debe ir cada pieza de la geometría del *sketch*, por lo que se considera un SBM, pero por el otro lado, cuando el usuario decide que primitiva geométrica tridimensional usar solo la

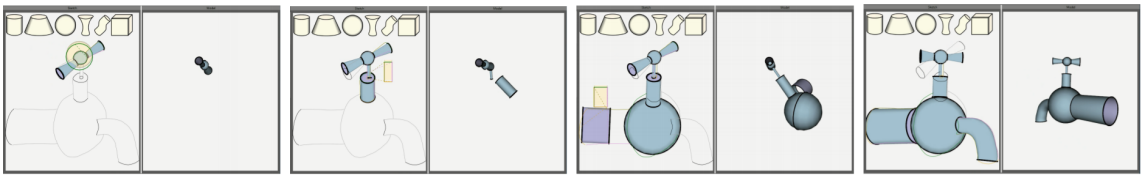


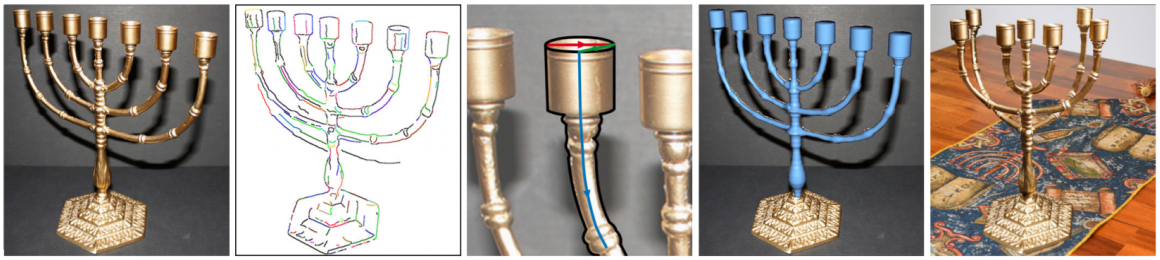
Figura 2.1: Pasos para la reconstrucción geométrica de un objeto 3D a partir de su *sketch*

suelta en el área deseada y el sistema debe decidir como ajustarla, por ello el sistema se clasifica como un SBR también. El segundo aporte destacable de esta obra es que las piezas geométricas no solo se ajustan al *sketch* en cada sección sin comunicación entre ellas, sino que obedecen una serie de reglas geosemánticas que proporcionan una coherencia al sistema que no es posible encontrar en otros trabajos. Como es posible deducir del párrafo anterior, la aportación más interesante de este trabajo se hace como SBR, ya que, aunque obvia una parte del proceso al dejarlo como una entrada del usuario (teniendo como premisa que la selección correcta de primitivas y su posición, aun es un problema difícil para una inteligencia artificial), el ajuste de primitivas y las reglas geosemánticas crean un modelo final de muy buena calidad. Para ajustar las primitivas tridimensionales Shtof propone que el *sketch* de entrada puede ser separados en sus trazos y estos a su vez pueden ser clasificados en dos categorías: los característicos (los cuales son independientes del ángulo de visión en 3D, como las curvas de una esfera) y los de silueta o borde (que dependen del ángulo de visión en 3D). Esta clasificación es hecha de manera semiautomática, pues el usuario marca una o dos curvas y el sistema hace la propagación necesaria. Mediante los trazos característicos podemos conocer los principales atributos de la primitiva tridimensional, pero con los trazos de contorno podemos hacer el ajuste de por optimización para obtener el modelo exacto que se ajusta a la forma, para esto consideramos que una región cerrada del *sketch* S se conforma de n puntos $s_1, s_2, \ldots, s_n$. Una vez que se conocen las curvas y trazos de un *sketch*, es posible posicionar una primitiva tridimensional, en el espacio seleccionado y tomando la premisa de que no existe ruido en el *sketch*, se hace el ajuste entre la ecuación paramétrica del modelo y las líneas del *sketch* mediante el método del Langragiano aumentado, en general la función objetivo cambia dependiendo del modelo paramétrico de primitiva tridimensional usado, pero de forma general nuestra función objetivo es: minimize $A_i = \sum_{i=0}^{N} w_i \left\|(F - s_i)\right\|^2$ subject to $G_k(A_1, \ldots, A_{i-1})$ con $k = 1, 2, \ldots, p$ Donde A_i es la i-esima primitiva que estamos ajustando, F es la ecuación paramétrica del modelo de primitiva tridimensional seleccionada, mientras w_i es un vector de pesos para las diferentes variables que conforman la ecuación paramétrica F. Cada que una nueva primitiva tridimensional es puesta en uno de los espacios del *sketch* debe ser ajustada al contenido actual, es decir, a todas las primitivas puestas anteriormente $A_1, \ldots, A_n$, mediante las restricciones geosemánticas del problema $G_1, \ldots, G_p$, es decir a condiciones de paralelismo y ortogonalidad dado que son figuras de orden ingenieril, la concentricidad y colinearidad de los centros tanto de las primitivas tridimensionales como de las curvas, así como de la coplanaridad de los planos que forman una primitiva tridimensional, como

para cada primitiva se aplican diferentes restricciones (obviamente para una esfera no podemos aplicar co-planaridad), p varia de modelo a modelo. El resultado del proceso completo es una figura que relaciona todas sus partes, de forma que no son solo una aproximación con figuras despegadas entre ellas, sino que sus caras, ángulos y demás relaciones, han sido optimizados con el fin de que la figura consiga una simetría y geometría similar a la de una pieza de ingeniería diseñada por el ser humano.

2.5.2 3-sweep: Extracting editable objects from a single photo

Trabajo propuesto por [Chen et al., 2013] basado en la publicación de [Shtof et al., 2013], pero que presenta una notable evolución respecto a este último, esta vez no se trata de otro trabajo que toma como entrada un *sketch* y luego lo categoriza de alguna forma particular antes de procesarlo, este es uno de los pocos trabajos que toman como entrada una fotografía RGB y la procesa para obtener una forma tipo *sketch*, mediante técnicas de extracción de bordes, aunque estas son más complicadas que las técnicas tradicionales, como suele ser la de [Canny, 1986], en vez de ello utiliza una técnica desarrollada por [Arbelaez et al., 2011] que permite la extracción jerárquica de bordes basada en *clustering* espectral, lo que permite la creación de bordes resistentes a los cambios de luz y sombra (aunque no así ante las texturas), al mismo tiempo que crea una estructura que relaciona los bordes entre ellos; el resultado de este proceso es una silueta resistente al ruido que puede categorizarse como el *sketch* de la fotografía. Ya construido el *sketch* de entrada, es necesaria la intervención del usuario de manera similar al sistema propuesto por Shtof, pero esta vez en lugar de soltar una primitiva tridimensional dentro de una zona del *sketch*, utilizaremos un sistema de tres trazos (de ahí el nombre de 3-sweep), los dos primeros marcaran la base de la primitiva tridimensional y el tercer trazo marca la dirección del mismo (el cual puede ser una recta o una curva), durante el tercer trazo el programa va rellenando automáticamente el *sketch* con una primitiva tridimensional, la cual en comparación del trabajo de Shtof, puede doblarse al tener un eje principal curvo, lo cual ayuda a construir un SBR robusto. El truco de poder hacer el trazado en tiempo real es hacerlo solo un tramo a la vez, pongamos como ejemplo la Figura 2.2, donde podemos ver el proceso que lleva la construcción de un modelo 3D utilizando el sistema 3-sweep, en este se traza la base de un cilindro pero como podemos ver se trata de un cilindro con un eje principal curvo, sin embargo, esta curvatura no afecta la orientación, por lo que es posible definir un primer círculo base como se muestra en el inciso c) de la Figura 2.2, se espera que el usuario usando el ratón dibuje una trayectoria similar a la que se puede ver en el mismo inciso en color azul, como el lector seguramente ya notó, el cilindro descrito no es regular sino que tiene severas deformaciones a lo largo de su trayectoria, lo cual al usar el método de ajuste de primitivas propuesto por Shtof, provocaría que el método del Langragiano aumentado solo se ajustara a estas deformaciones, dando como resultado un cilindro curvo pero liso. Para resolver esta eventualidad, Chen propone usar un sistema de múltiples instancias de la misma



(a) (b) (c) (d) (e)

Figura 2.2: Procedimiento de 3-sweep para crear un modelo geométrico tridimensional a partir de un *sketch*

primitiva: $A_1, A_2 \ldots A_n$, cada una de las cuales se ajusta rápidamente al *sketch* siguiendo el eje principal E formado por una serie de puntos $e_1, e_2, \ldots, e_n$ marcado por el usuario, al ser primitivas sumamente delgadas abarcan solo una pequeña porción del *sketch* por lo que el ajuste no requiere de tantos puntos de comparación con el *sketch*, que se considera formado por una serie de trazos, cada trazo a su vez se forma de una serie de puntos $p_1, p_2, \ldots, p_n$, el proceso que hace destacar el trabajo de Chen es la premisa de que no es necesario hacer la optimización del ajuste de la primitiva tridimensional de forma exhaustiva cada que el usuario avanza a lo largo de E, sino que cada que se obtiene un nuevo punto e_i crearemos una nueva primitiva A_i la cual copia los parámetros de la primitiva A_{i-1} y los rota basado en la diferencia entre los ángulos de las rectas $overlinee_ie_{i-1}$ y $overlinee_{i-1}e_{i-2}$, esto con la finalidad de tener una buena aproximación para comenzar el ajuste de la primitiva tridimensional; conociéndose de antemano que también solo se podrá ajustar a puntos del trazo del *sketch* posteriores a p_k, donde p_k es el último punto del *sketch* donde A_i se ajustó a ese trazo. El resultado es una serie de primitivas tridimensionales continuas tales que cada una puede ajustarse a diferentes grosores de la forma, lo que da un resultado bastante estético como puede apreciarse en la Figura 2.2 inciso d).

La representación descrita en el párrafo anterior aún es muy complicada por lo que es necesario optimizar la forma para ayudar a que la mayoría de las formas se adapten a unas pocas primitivas tridimensionales, para ello, de cada una de las primitivas instanciadas debemos conocer sus puntos característicos (en un cuboide es la esquina, en un cilindro los centros etc.) $C_{i,j}$, siendo i la i-ésima primitiva instanciada y j el j-ésimo punto característico, de forma que podamos formar una función objetivo como la que sigue: $minimize\ A = \sum_{i=1}^{n} w_i(\sum_{j=1}^{m_i} \|C_{i,j} - F_{i,j}(z_i)\|^2)\ subject\ to\ G_k(C_{1,1}, ..., C_{n,m_n})$ con $k = 1, ..., p$ Donde m_i es el número de ejes de la primitiva tridimensional analizada, w_i es un peso propuesto por Chen que se calcula tomando en consideración la base y el largo de la forma [Chen et al., 2013]. $F_{i,j}$ es una función que representa la posición en la que debería estar el punto característico $C_{i,j}$ pero en lugar de definirlo en función de los 3 ejes, solo usamos la profundidad z_i, lo cual es posible gracias al comportamiento estable de las primitivas tridimensionales. Toda la función objetivo está sujeta a p restricciones G_k, las cuales son las mismas restricciones geosemánticas dictadas por Shtof. Al final obtendremos una representación del objeto con solo unas

pocas primitivas que se va construyendo en tiempo real a partir de lo que el usuario percibe como una foto, pero que es tratado como un *sketch* internamente para mejorar el rendimiento de la reconstrucción.

2.5.3 How do humans sketch objects?

Uno de los trabajos contemporáneos más representativos en clasificación de *sketches*, presentado por [Eitz et al., 2012], como un extenso trabajo sobre las diferencias actuales entre la clasificación que los humanos hacen de los *sketches* y la hasta ahora alcanzada por las computadora siguiendo la tendencia de usar descriptores de imágenes de diferentes secciones del *sketch* para formar vectores característicos que después pudieran ser comparados con una base de conocimiento. Este trabajo proporciona un banco de datos procesados de *sketches* muy amplio, bajo la premisa de ofrecer datos de 250 categorías de objetos de uso común, para saber que categorías son las más usuales en la vida diaria se hace referencia a la base de etiquetado propuesta por [Russell et al., 2008], que contiene las 1000 etiquetas más frecuentemente utilizadas por las personas al referirse a objetos del mundo cotidiano, la base es analizada para agrupar categorías duplicadas o similares (e.g. carro, carro de lado, carro por atrás etc.), lo que da como resultado 250 categorías únicas. Usando la herramienta *Amazon Mechanical Turk* (AMT) se solicita a 90 participantes que hagan el trazado y etiquetado de 250 dibujos tipo *sketch*, después de quitar los errores claros, se conforma un banco de *sketches* de 20,000 imágenes. El tamaño de este banco de datos permite un correcto acercamiento al problema actual de la clasificación de *sketches*, el cual como se trató en la sección 2.3, consiste en clasificar bancos de *sketches* con una gran cantidad de categorías (¿50). El enfoque de construcción del vector característico consiste en tratar cada imagen de un *sketch* por medio de un descriptor de imágenes, en este caso el *sketch* S se muestra como una colección de píxeles tal que $S \in \mathbb{R}^{m \times n}$, lo cual para fines de clasificación por medio de descriptores es algo impráctico por lo que los autores proponen mapear S a un espacio de 256×256 píxeles, como se desea obtener todo un vector característico y no solo el gradiente de todo el mapa de bits, se usara el esquema de *bag-of-features* [Sivic et al., 2003], en el que se dividirá esta nueva imagen en cuadros con centros u, v de 28×28 píxeles hasta obtener un total de 784 cuadros, de ellos se computará el gradiente local representado por $g_{u,v} = \nabla S$ y su orientación $O \in [0, \pi)$, acomodaremos el valor de $\|g\|$ en r contenedores direccionales de acuerdo al valor de O, para obtener un descriptor de orientación l, Eitz propone un valor de $r = 4$, esto nos generara un pequeño histograma que tenemos que ponderar con los vecinos adyacentes para suavizar los valores y evitar cambios bruscos en los bordes, este procesamiento nos dará como resultado un vector $d = [l_1, ..., l_r]^T$, que debemos normalizar tal que $\|d\|_2 = 1$. Esta representación es compactada utilizando un vocabulario visual mediante *K-mean clustering*, además como el lector puede darse cuenta, el proceso anteriormente descrito no difiere mucho del famoso descriptor SIFT [Lowe, 1999], la única diferencia es que el descriptor propuesto por Eitz, solo entrega valores dependientes de la orientación. Después del compactado y de un análisis

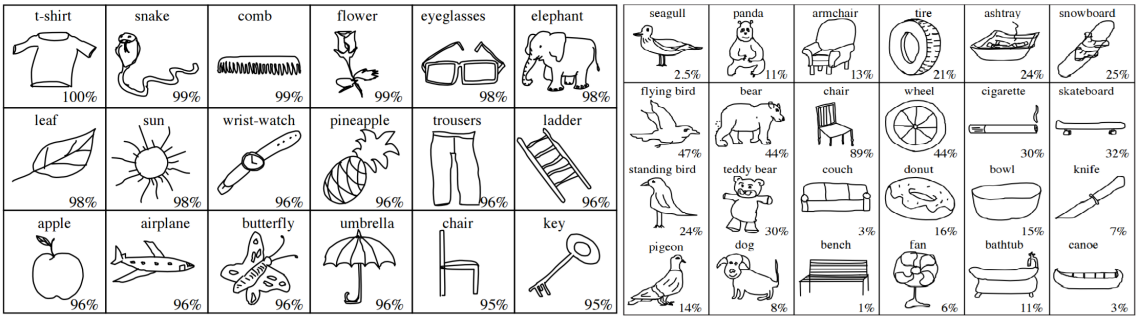


Figura 2.3: A la derecha los *sketches* que son fácilmente reconocibles, a la izquierda *sketches* con los índices de asertividad más bajos

PCA tenemos un vector característico con una cardinalidad de 500 por cada *sketch*, mientras tenemos 20,000 *sketches*, lo que forma un banco de datos bastante robusto, la metodología de clasificación se lleva por dos vías, la primera por los clasificadores K-NN y la segunda mediante las máquinas de soporte vectorial, debido a la magnitud de la base de datos los experimentos suelen llevar un tiempo considerable, por lo este articulo solo trata con estos dos enfoques, aunque artículos más recientes hacen uso de diferentes tipos de redes neuronales artificiales. En ambas metodologías se utilizan dos enfoques, uno suave y uno fuerte, que hacen referencia al poder de computo necesario para el vocabulario visual. En el caso de KNN se exploran valores de $K \in [1, 2, \ldots, 5]$ y las distancias Manhattan, euclidiana, de cosenos y de correlación, los resultados de combinar los diferentes valores de K con las diferentes distancias, nos arrojaron que el mejor resultado para el perfil suave del vocabulario visual, es un valor de $K = 4$ con la distancia Manhattan, obteniendo resultados del 44.5 % de asertividad, mientras que para el perfil duro los mejores resultados se obtienen con $K = 5$ y la distancia del coseno, pero obteniendo un valor de apenas el 38 % de asertividad; ambas pruebas y las posteriores se realizaron usando *3-fold cross validation*. Las siguientes pruebas se realizarían con máquinas de soporte vectorial, enfocándose en el kernel gaussiano con $\gamma \in [10, 20, ..., 100]$ y $C \in [1, 2, ..., 100]$, presentando una combinatoria más amplia, pero que al final se resuelve en un resultado del 54 % de asertividad para el paradigma duro con valores de $\gamma = 17.8$ y $C = 10$, mientras que para el paradigma suave las variables se repiten pero esta vez con resultados del 56 % de asertividad.

La explicación que el autor ofrece para estos índices de asertividad es el hecho de que muchos de los trazos que conforman el *sketch* de un objeto, se repiten para representar otro objeto, tomemos como ejemplo la Figura 2.3, donde Eitz nos presenta los diferentes índices de asertividad entre diferentes *sketches*, a la derecha de la Figura 2.3 notamos figuras con trazos irrepetibles y que tienen altos índices de asertividad, mientras a la izquierda vemos *sketches* que tienen un índice de asertividad muy bajo, e.g., siguiendo el proceso de creación de los descriptores locales, sería muy fácil confundirse entre un tazón, una tina y una canoa, pues muchos de los gradientes locales serían muy similares.

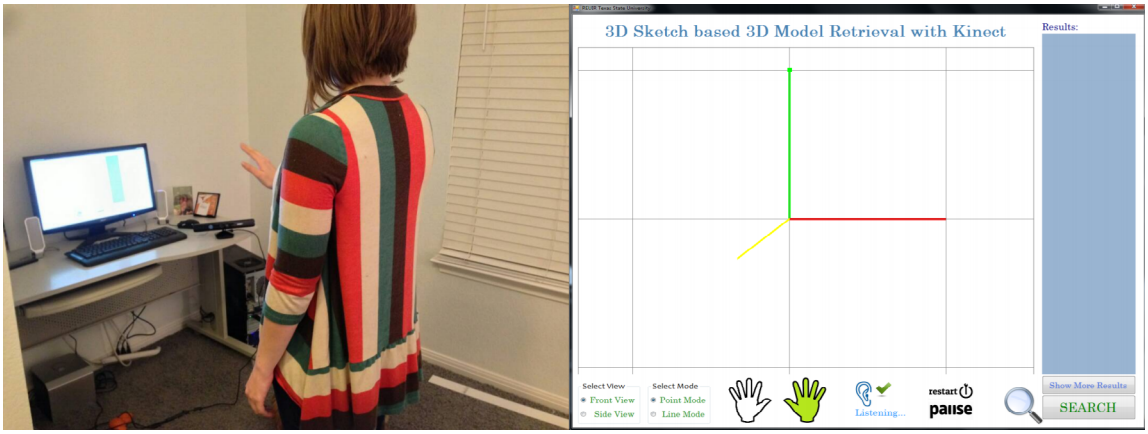


Figura 2.4: A la derecha un usuario dibujando un *sketch* en el aire en tres dimensiones. A la izquierda el programa de captura de datos

2.5.4 3D Sketch Recognition Using the Microsoft Kinect

Tesis de la Universidad de Texas, desarrollada por [Bulgerin, 2014], que sienta el precedente de la casi nula existencia de trabajos enfocados al *sketching* en tres dimensiones, pero como agregar una dimensión más al *sketching* puede traer consigo beneficios importantes, logrando aumento de hasta el 20 % en la asertividad, comparado contra la misma técnica aplicada para imágenes en dos dimensiones. Para captar el *sketch* en tres dimensiones, Bulgerin hace uso del Microsoft Kinect® y de su API de desarrollo, con lo que crea un programa de detección en el que el usuario, usando las manos, crea un *sketch* en el aire, tal cual lo muestra la Figura 2.4. En el trabajo participan 10 usuarios capturando 70 *sketches* individuales lo que nos da un banco de datos de 700 patrones, lo cual nos permite trabajar adecuadamente. El procedimiento seguido por Bulgerin ha sido muchas veces estudiado y modificación para la reconstrucción y reconocimiento de *sketches* en dos dimensiones, con resultados poco prometedores, pero el verdadero aporte de este trabajo radica en la propuesta metodológica e implementación del mismo concepto en tres dimensiones, lo que nos muestra como un paradigma que es conocido por su índice de asertividad moderado, puede obtener mejores resultados al agregar más información de la dimensionalidad. Este trabajo se basa en los descriptores HOG [Dalal and Triggs, 2005], retomando el trabajo y el banco de *sketches* de [Eitz et al., 2012], que recordaremos posee 250 clases de objetos diferentes, aunque Bulgerin recorta el banco de datos para solo trabajar con 70 diferentes clases, la metodología es muy similar a la descrita por Eitz, con la diferencia de que ya que Eitz aplica un descriptor SIFT que solo utiliza orientaciones, Bulgerin propone utilizar HOG, que ya se basa en el histograma y la orientación como únicas salidas, haciendo un ajuste para acomodar el valor de $\|g\|$ en r en 8 contenedores de acuerdo a la orientación O, en lugar de hacerlo en 4 como describe el trabajo original de Eitz, este cambio se ve influenciado por un manejo más suave en los cambios de dirección. En lugar de crear cuadros dentro de la imagen 2D, Bulgerin

crea cubos o *voxels* de tamaño $n \times n \times n$ a partir de la normalización del dibujo original en tres dimensiones, proponiendo $n = 64$ como un valor adecuado. Cada *voxel* es marcado como cero o uno dependiendo si el *voxel* posee una parte del trazado o no, esto agiliza el computo pues en tres dimensiones se dan una gran cantidad de *voxels* vacíos, con este enfoque binario podemos obtener muy fácilmente el gradiente de la siguiente manera:

$$\begin{aligned}\nabla f(x,y,z) &= (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}) \\ &= f(x+1,y,z) - f(x-1,y,z) + \\ &\quad f(x,y+1,z) - f(x,y-1,z) + \\ &\quad f(x,y,z+1) - f(x,y,z-1)\end{aligned}$$

Este valor puede ser fácilmente representado en coordenadas esféricas bajo el siguiente esquema:

$$\begin{pmatrix} \theta \\ \phi \\ r \end{pmatrix} = \begin{pmatrix} arccos(\frac{z}{\sqrt{x^2+y^2+z^2}}) \\ atan2(x,y) \\ \sqrt{x^2+y^2+z^2} \end{pmatrix}$$

Con los valores de θ y ϕ se forma un histograma dos-dimensional el cual es normalizado y concatenado a todos los histogramas de los *voxels*, lo que al final nos da el vector característico del *sketch*. Siguiendo con el protocolo propuesto por Eitz, se utilizan máquinas de soporte vectorial con un kernel gaussiano, al cual se varían los parámetros $\gamma \in [10^{-4}, 10^{-3}, \ldots, 10^{2}]$ y $C \in [1, 2, ..., 100]$ obteniendo resultados del 59 % de asertividad para las 70 clases propuestas con $\gamma = 10^{-4}$ y $C = 10$, después el experimento se repetiría bajo la misma metodología pero adquiriendo los *sketches* en 2D, lo que llevo a resultados de asertividad máxima de 33 %, lo que demostró la clara diferencia al adquirir un *sketch* en una dimensión mayor, a diferencia del trabajo de Eitz, se utiliza el método de *Leave One Out* para validar estos resultados.

Capítulo 3

Adquisición de imágenes

Uno de los diferenciadores de este trabajo respecto a otros, en el estado del arte, es el uso de datos de profundidad. Para esto, es necesaria una cámara RGB-D, que entrega una imagen fotográfica RGB, la cual es un mapa de píxeles con coordenadas x, y, donde a cada ubicación le corresponde una combinación de colores en rojo, verde y azul (*Red*, *Green* y *Blue* que en inglés forman las siglas RGB). Junto con la imagen fotográfica, se entrega un mapa de profundidad con coordenadas x, y donde a cada ubicación le corresponde una distancia o valor de profundidad.

Las imágenes RGB se obtienen a partir de un sensor CMOS. Sin embargo, las imágenes de profundidad pueden ser obtenidas por diferentes tecnologías, entre las que se encuentran:

- *Structured Light*. Mide la distancia con un objeto tridimensional a partir de las deformaciones que sufre un patrón proyectado. Este patrón es una imagen con ciertas características que se proyecta en luz infrarroja y que es invisible al ser humano.
- *Time of Flight*. La medición de la distancia se hace al emitir una señal de luz y luego medir el tiempo que tarda en llegar a su objetivo. Esto se repite emitiendo pulsos de luz para cada pixel de la imagen.
- Laser. Proyecta un punto laser sobre una superficie utilizando, al menos, dos galvanómetros; el punto es registrado por un espejo.

En particular, nuestra cámara RGB-D utiliza tecnología de luz estructurada (*Structured Light*), la cual será explicada en la Sección 3.1.

Un problema no menospreciable es que, a diferencia de las cámaras RGB, las cámaras RGB-D no entregan un resultado único. De hecho, se emplea el término "cámara RGB-D" para referirse, en realidad, a un arreglo de, al menos, dos cámaras, un sensor CMOS para fotografías y una cámara de profundidad. Esto tiene como consecuencia que no se puede acceder a los valores de color y profundidad, únicamente mediante las coordenadas x, y, debido a que la imagen RGB está separada de la imagen de profundidad. Además, ambas cámaras se encuentras espacialmente separadas y entregan

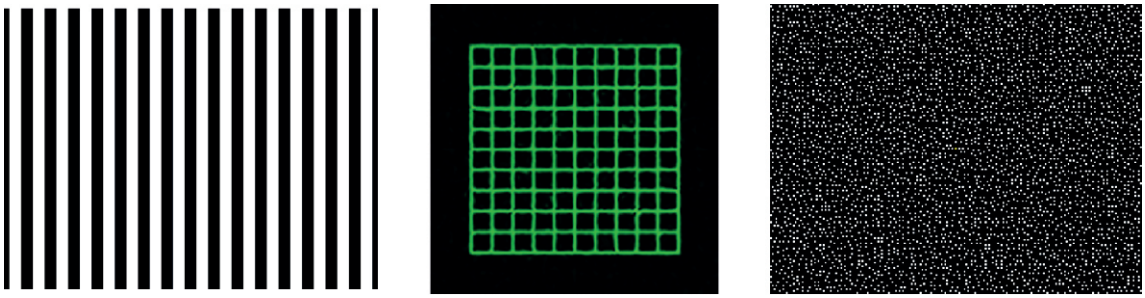


Figura 3.1: Diferentes tipos de patrones utilizados por las cámaras RGB-D que emplean la técnica de luz estructurada

imágenes de diferente tamaño y relación de aspecto, problema que será solucionado en la Sección 3.3, haciendo uso de técnicas propias de calibración de cámaras.

Otro de los problemas comunes al tratar con imágenes RGB-D son las luces y sombras provocadas por frecuencias infrarrojas, ajenas a la cámara RGB, o por obstrucciones. Si bien estas no son iguales a sus pares en el espacio RGB, si pueden volverse un problema que afecte las mediciones. Afortunadamente existen técnicas que permiten tratar con las sombras, lo cual será tratado más a detalle en la Sección 3.2.

3.1 Cámara de profundidad

En la presente tesis, se utiliza un dispositivo Microsoft Kinect® como cámara RGB-D en su primera versión, la cual trabaja con luz estructurada, i.e., proyecta un patrón infrarrojo invisible al ojo humano que es captado por una cámara infrarroja. La diferencia entre lo que sabe que se debe proyectar y la proyección obtenida da como resultado el cálculo de la distancia. Es importante hacer notar que esta técnica depende de la fidelidad con la que la luz llega a un sólido para formar el patrón. Por lo tanto, esta técnica es afectada por interferencia infrarroja, como la luz solar extrema u otros proyectores infrarrojos, así como por la refracción del medio, lo que la hace poco apta para trabajar bajo el agua o con otros líquidos.

Existen muchos patrones que pueden ser proyectados para calcular la profundidad, como patrones lineales, de tipo ajedrez, cuadriculados y de dispersión de puntos (ver Figura 3.1), siendo estos últimos los proyectados por Kinect en la versión que atiende esta tesis. Los puntos son reconocidos en secciones, lo que ayuda a que la información sea mapeada a una imagen de hasta 640 x 480 píxeles (aunque el modo de 320 x 240 píxeles suele ser más estable), i.e., cada sección representará un píxel de la imagen de profundidad. Cada una de las secciones puede ser concebida como un rectángulo, con un conjunto de puntos infrarrojos, que puede ser fácilmente diferenciado de los otros rectángulos, mediante un código adscrito en las posiciones de cada conjunto de puntos perteneciente a cada rectángulo. Esto es más ampliamente descrito en las patentes de Kinect US2010/0290698, US2010/0118123 y US2010/020078.

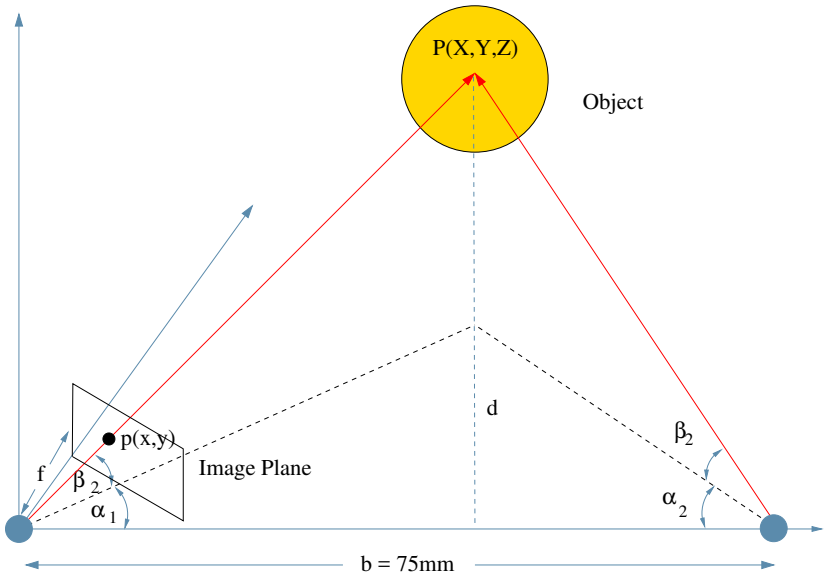


Figura 3.2: Sistema de triangulación que permite conocer la distancia entre una cámara RGB-D y el objeto de interés

Una vez que el patrón de puntos es reconocido como una sección, se debe saber cuál es la profundidad. Es en este punto donde nuestro conocimiento del sistema ayudará a resolver un sistema por triangulación. En general, la cámara y el proyector infrarrojo están alineados en los ejes Y y Z, pero en lo que respecta al eje X hay una separación de 75mm. Ambos están alineados a 90 grados respecto a la horizontal principal del producto y a 0 grados respecto a los otros dos ejes. Estos valores son buenas aproximaciones dictadas por el diseño de Kinect, pero pueden variar ligeramente debido a errores de fabricación. Dado que estos valores son conocidos, es posible resolver el sistema por triangulación que se muestra en la Figura 3.2.

El sistema mostrado en la Figura 3.2 parece complicado de resolver, pero puede descomponerse en dos sub-problemas más sencillos, cuyos datos esenciales pueden obtenerse a partir de mediciones. Primero, se puede descomponer el problema considerando únicamente la distancia d entre el objeto y el plano horizontal, creado por la cámara y el proyector infrarrojo (ver la sección (a) de la Figura 3.3), resolviendo los ángulos α_1 y $alpha_2$ a partir de la diferencia horizontal entre el ángulo de proyección y el captado por la cámara infrarroja y resolviendo d mediante la ley de senos, como se muestra en la Ecuación 3.1:

$$d = \frac{b \times sin(\alpha_1) \times sin(\alpha_2)}{sin(180 - \alpha_1 - \alpha_2)} \tag{3.1}$$

Para resolver la distancia tridimensional D, se toma el sistema de la sección (b) de la Figura 3.3, que se basa en la distancia d obtenida con anterioridad. El ángulo β_2 es el mismo desde la proyección que desde la cámara, ya que ambos dispositivos están

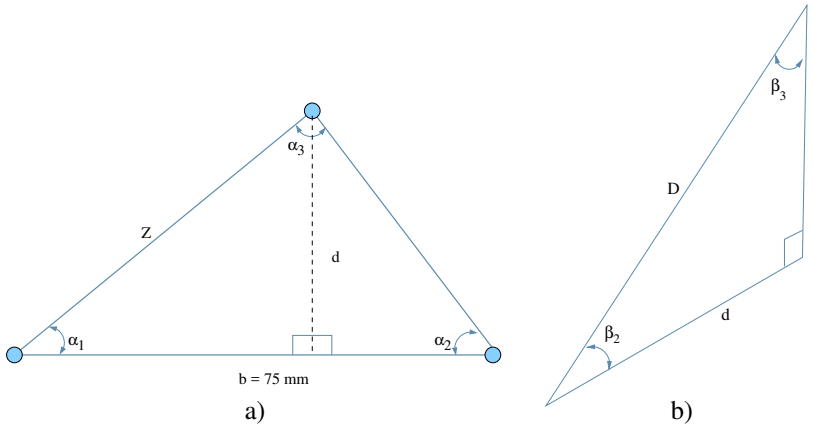


Figura 3.3: Descomposición del problema de triangulación, en dos problemas de triangulación más sencillos: a) triangulación horizontal y b) triangulación vertical

alineados en la vertical, por lo que β_2 puede ser deducido de la medición de la cámara y del ángulo de proyección del proyector. Finalmente, el ángulo $\beta_1 = 90$, así que D puede ser calculada según lo estipula la Ecuación 3.2, utilizando nuevamente la ley de senos:

$$D = \frac{d}{sin(90 - \beta_2)} \tag{3.2}$$

El valor de profundidad D entre Kinect y un objeto está representado en milímetros. Sin embargo, por cuestiones de almacenaje, este valor es normalizado a un espacio de 0 a 255 para que pueda ser guardado en un byte y, a su vez, representado como una imagen en escala de grises, siendo 0 el valor del corte cercano (por defecto 800mm) y 255 el valor del corte lejano (3975 mm por defecto), aunque estos valores son configurables. De esta manera, en el modo por defecto, los 256 valores posibles representados por Kinect describen 12.4 milímetros cada uno, pero queda a consideración del usuario cambiar los niveles de profundidad.

3.2 Inpainting

El cálculo desglosado en la Sección 3.1 se repite para cada píxel de la imagen de profundidad, lo que dará como resultado algo similar a lo que se muestra en la Figura 3.4. Esto ya sería bastante útil excepto por el problema de las sombras infrarrojas provocadas por la obstrucción del patrón proyectado, dejando a la imagen con huecos negros (valores nulos). Así que cuando se solicite la información de profundidad de ese píxel específico, el valor obtenido será distinto al valor real de profundidad. Gracias a la naturaleza de las sombras infrarrojas, este problema puede tratarse en forma sencilla mediante técnicas de *inpainting*.

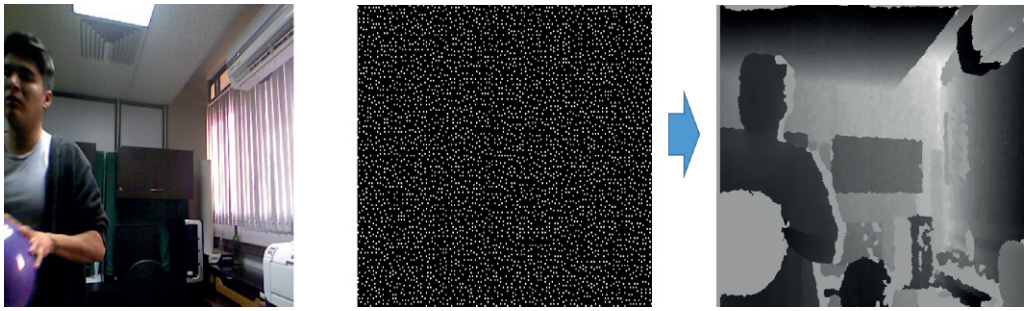


Figura 3.4: A la izquierda se muestra la imagen RGB, al centro el patrón aplicado y a la derecha la imagen de profundidad correspondiente sin tratar

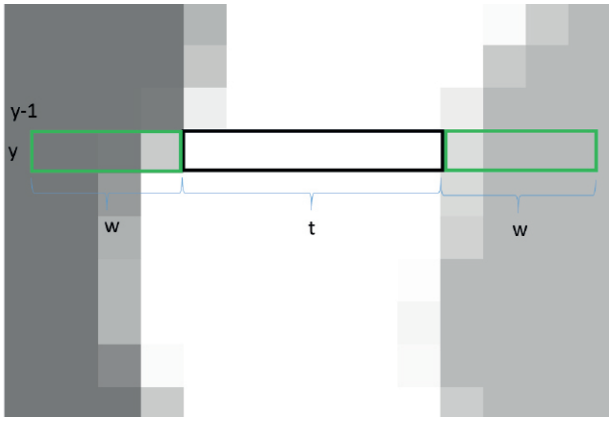


Figura 3.5: Descomposición del hueco infrarrojo en un problema 1-dimensional de tamaño $t + 2w$, siendo w las anclas de la ventana t hacia un espacio de mediciones reales

Los algoritmos de *inpainting* no sólo se usan para imágenes de profundidad, sino también en gran variedad de procesos en el tratamiento digital de imágenes, por lo que existen muchas técnicas reportadas en la literatura. En esta tesis, nos centraremos en uno de los algoritmos, computacionalmente más económicos, descrito por [Joachimiak et al., 2013], el cual llena los agujeros con base en la información más cercana.

De acuerdo con esta técnica, el hueco es procesado horizontalmente, línea por línea, en una ventana 1-dimensional, i.e., de un píxel de altura y de $t + 2 \cdot w$ píxeles de largo, donde t es el largo del hueco en esa línea y w es una extensión de la ventana. De este modo, se pueden conocer los valores de los píxeles vecinos a la izquierda y a la derecha, tal como se observa en la Figura 3.5.

La técnica de *inpainting* consiste en dos pasos. En el primero, se necesita calcular la distancia máxima Z_{max} para la actual fila del hueco, considerando las Ecuaciones 3.3 y 3.4:

$$Z_{max}(t, y) = (1 - k) \cdot Z_{av}(t - w...t + w, y) + k \cdot Z_{max}(t, y - 1) \tag{3.3}$$

Figura 3.6: Resultado del proceso de *inpainting*: a la izquierda se muestra la imagen original y a la derecha la imagen después del proceso de *inpainting*

$$k = \frac{1}{1 + |Z_{max}(t, y-1) - Z_{av}(t-w...t+w, y-1)|} \tag{3.4}$$

El primer parámetro, tanto de Z_{max} como de Z_{av}, es un rango de píxeles, ya que ambas operaciones buscan sobre la fila indicada por el segundo parámetro, pero sólo en el rango indicado por el primer parámetro. Mientras Z_{max} busca la profundidad máxima de un rango, Z_{av} busca el promedio de las profundidades en un rango.

El segundo paso de la técnica de *inpainting* es decidir cuál de los valores de profundidad debe ser colocado en el rango t. Para ello, se debe encontrar $Z_{cand} \in Z_l, Z, r$, de tal manera que minimice la distancia marcada por 3.5, donde Z_l representa el valor de profundidad del límite izquierdo de t y Z_r es el homónimo derecho.

$$|Z_{max} - Z_{cand}| < T \tag{3.5}$$

Se escoge Z_{cand} como el valor para rellenar el rango t, tal que minimice la distancia y que cumpla la Inecuación 3.5, donde T es un valor propuesto para evitar cubrir huecos que tienen que estar ahí debido a los cortes cercanos. Puede ser establecido con un valor de 800mm en caso de Kinect, pero este valor varía entre diferentes modelos de cámaras de profundidad. En caso de no estar disponibles los valores Z_l o Z_r, se toma Z_{max} como valor para rellenar t, lo mismo pasa en caso de no cumplirse 3.5. Este proceso se repite para cada fila que compone al hueco provocado por una sombra infrarroja y el resultado es una imagen mucho más suave y tratable como se puede ver en la Figura 3.6.

3.3 Matriz de transformación

El proceso hasta ahora descrito, en este capítulo, ha dado como resultado una imagen de profundidad con la que es posible trabajar, mientras que la obtención de la imagen RGB es trivial, por lo que parecería que se tienen los elementos de entrada necesarios para comenzar la extracción del *sketch*. Sin embargo, esto no es del todo cierto ya que, aunque no se necesita otro elemento de entrada, si se requiere una manera de pasar del espacio RGB al espacio de profundidad y viceversa. Esto quiere decir que, dadas las coordenadas de un píxel RGB, se obtenga la profundidad de ese píxel o que, dadas las coordenadas en el mapa de profundidad, se obtenga su código RGB (ver Figura 3.7). Recordando algunas ideas básicas, se puede ver que es tan simple como usar la misma coordenada de ambos pares al escalar las imágenes. De hecho, las imágenes RGB y de profundidad no tienen el mismo tamaño ni la misma relación de aspecto; tampoco son tomadas desde el mismo punto (la cámara RGB y la cámara de profundidad están separadas) y, además, los pequeños errores de fabricación influyen en la rotación entre imágenes.

Para resolver este problema de emparejamiento de imágenes, se utiliza una matriz de transformación, de manera muy similar a lo que sucede en la calibración de cámaras en técnicas estereoscópicas. Sin embargo, en el caso de las imágenes RGB-D, se tiene una problemática extra. En técnicas clásicas para imágenes estereoscópicas, todas las imágenes trabajan en el espacio RGB, por lo que se pueden encontrar fácilmente puntos característicos que existan en todas las imágenes, y que permitan emparejarlas y encontrar una matriz de transformación entre ellas. Cuando se tienen imágenes que trabajan en espacios distintos, es necesario encontrar puntos que sean invariantes en ambos espacios, lo cual es complicado *per se*, ya que se han elegido estos dos espacios por las diferencias complementarias entre ellos.

Es posible resolver este problema buscando los parámetros intrínsecos y extrínsecos de la transformación y formando una matriz que permita, dadas las coordenadas de un espacio, obtener las coordenadas correspondientes de otro espacio. Los parámetros extrínsecos son parámetros externos al sensor y definen la posición t y orientación R del sensor respecto al plano que la cámara RGB y la de profundidad definan. Estos parámetros son ejemplificados en la sección (a) de la Figura 3.8. Los parámetros intrínsecos, por otro lado, hacen referencia a las propiedades internas del sensor, i.e., la longitud del foco (f_x y f_y), las coordenadas del punto principal (c_x y c_y) y la oblicuidad de ambos ejes de la imagen (α). Estos parámetros pueden ser visualizados en la sección (b) de la Figura 3.8.

La transformación completa entre espacios viene dada por la multiplicación matricial de los parámetros intrínsecos y extrínsecos, según la Ecuación 3.6:

$$sp = K[Rt]P \tag{3.6}$$

Donde K es la matriz de parámetros intrínsecos formada de la siguiente manera:

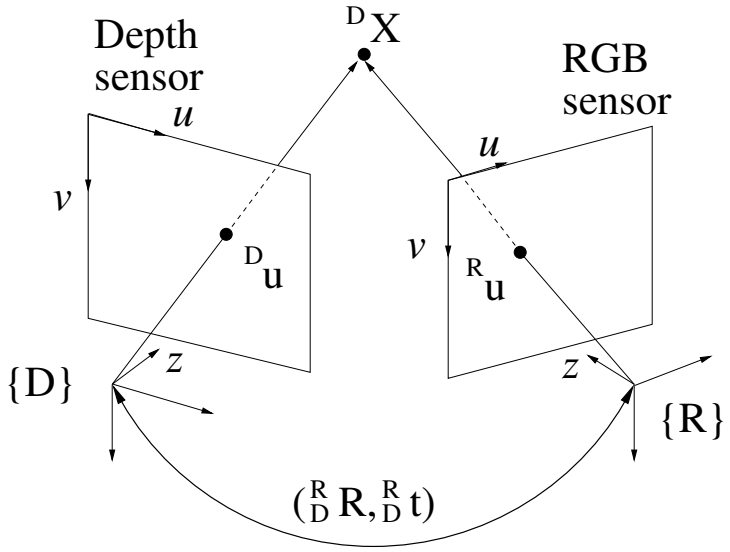


Figura 3.7: Esquema básico de la matriz de transformación entre el espacio RGB y el espacio de profundidad

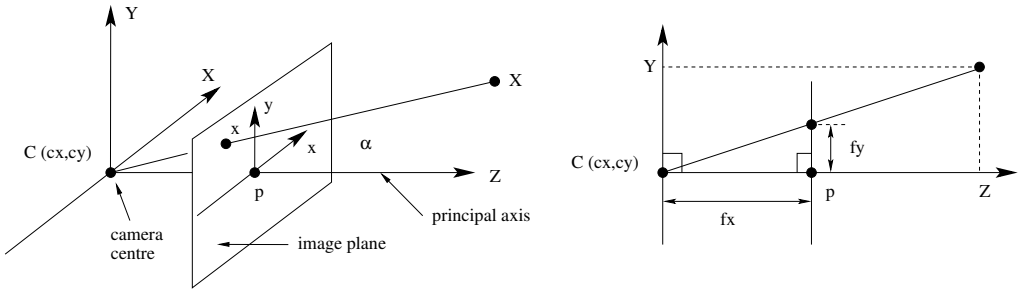


Figura 3.8: Parámetros internos y externos de la cámara: (a) muestra la representación gráfica de los parámetros extrínsecos, mientras que (b) muestra los parámetros intrínsecos

$$K = \begin{bmatrix} f_x & \alpha & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

R es la matriz de rotación compuesta por los vectores ortogonales r_i, cada uno de los cuales corresponde a la rotación tridimensional en uno de los ejes y está formado por la triada $(r_{i,1}, r_{i,2}, r_{i,3})$, dando como resultado la siguiente matriz:

$$R = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,1} & r_{3,2} & r_{3,3} \end{bmatrix}$$

El vector t se refiere a la posición y es sólo una triada con los valores de traslación en cada eje:

$$t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

El punto p es un punto en el espacio de profundidad, el punto P es un punto en el espacio RGB y s es un factor de escalamiento.

Como se puede apreciar de la definición de la Ecuación 3.6, podría parecer muy difícil encontrar tantas variables, pero la literatura en métodos de calibración de cámaras es extensa y se pueden ajustar estos métodos para encontrar la matriz de transformación. Usualmente estas técnicas iteran sobre un conjunto de puntos aportados por una imagen que contiene un patrón, el cual es capturado por las dos cámaras a la vez; puntos específicos de esa imagen son captados por ambas cámaras y entonces se inicia un proceso de minimización del error (mínimos cuadrados, RANSAC, metaheurísticas u otras). Uno de los patrones más usuales es el patrón de tablero de ajedrez, introducido por [Zhang, 2000], el cual es sencillo de usar porque toma como puntos característicos las esquinas de un tablero de ajedrez que puede ser impreso por cualquier impresora (ver Figura 3.9).

Desafortunadamente este paradigma no funciona de la misma manera para imágenes de profundidad. Obviamente lo único que se verá del patrón de tablero de ajedrez será el rectángulo que representa a la hoja de papel, como se aprecia en la Figura 3.10. Por lo tanto, el patrón de ajedrez o cualquier otra textura impresa no puede ser utilizado para la calibración. El mismo Zhang publicaría más tarde una actualización de su método en el que era posible usar cámaras de profundidad [Zhang and Zhang, 2011]. En este método, el usuario señalaba manualmente una serie de esquinas en la imagen

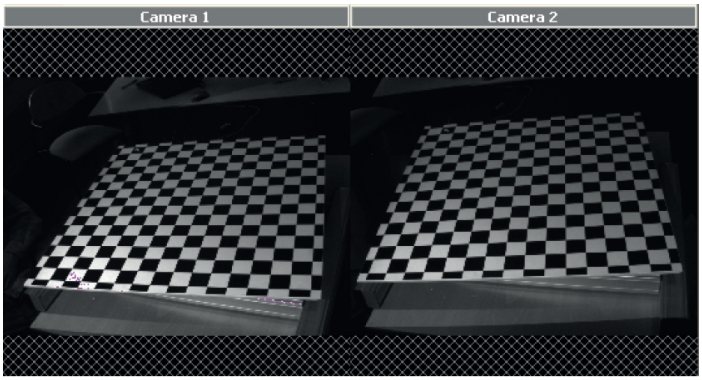


Figura 3.9: Patrón de tablero de ajedrez en dos imágenes tomadas por cámaras distintas y la localización de las esquinas dentro del patrón

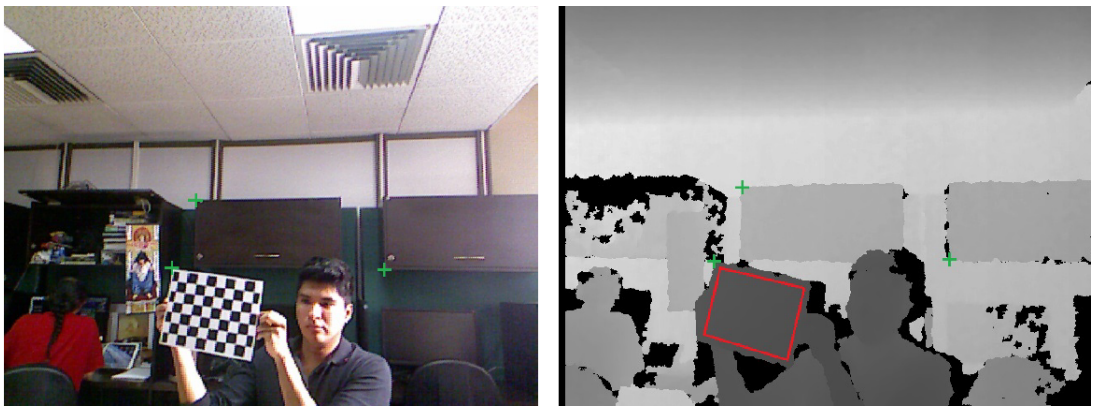


Figura 3.10: Captura, tanto en la cámara RGB como en la cámara de profundidad, del patrón de tablero de ajedrez y el posicionamiento manual de esquinas en objetos aleatorios identificados por el usuario

RGB y buscaba las mismas esquinas en la imagen de profundidad, como se muestra en la Figura 3.10. Como el lector podrá imaginarse, este termina siendo un problema tedioso para el usuario y depende, en gran medida, de su pulso para señalar adecuadamente el mismo punto en ambas imágenes. Si bien podría pensarse en un proceso de detección de esquinas para automatizar el proceso, lo cierto es que la misma Figura 3.10 da la negativa a esta propuesta, pues en la imagen de profundidad muchas esquinas son suavizadas y aparecen muchas otras, producto del ruido.

Un método más robusto al ruido y a la falta de texturas para obtener la matriz de transformación, fue propuesto por [Staranowicz et al., 2013]. Este método se basa en la observación de que una esfera será representada como un círculo, tanto en la imagen RGB como en la imagen de profundidad, y su contorno puede ser utilizado como el conjunto de puntos de entrada de la calibración, supliendo a las esquinas del patrón de ajedrez o de cualquier otro. De esta manera, sólo hace falta un conjunto de imágenes de una esfera que pase frente a ambas cámaras, sin que el usuario deba ingresar manualmente ninguna coordenada, como se aprecia en la Figura 3.11. Las

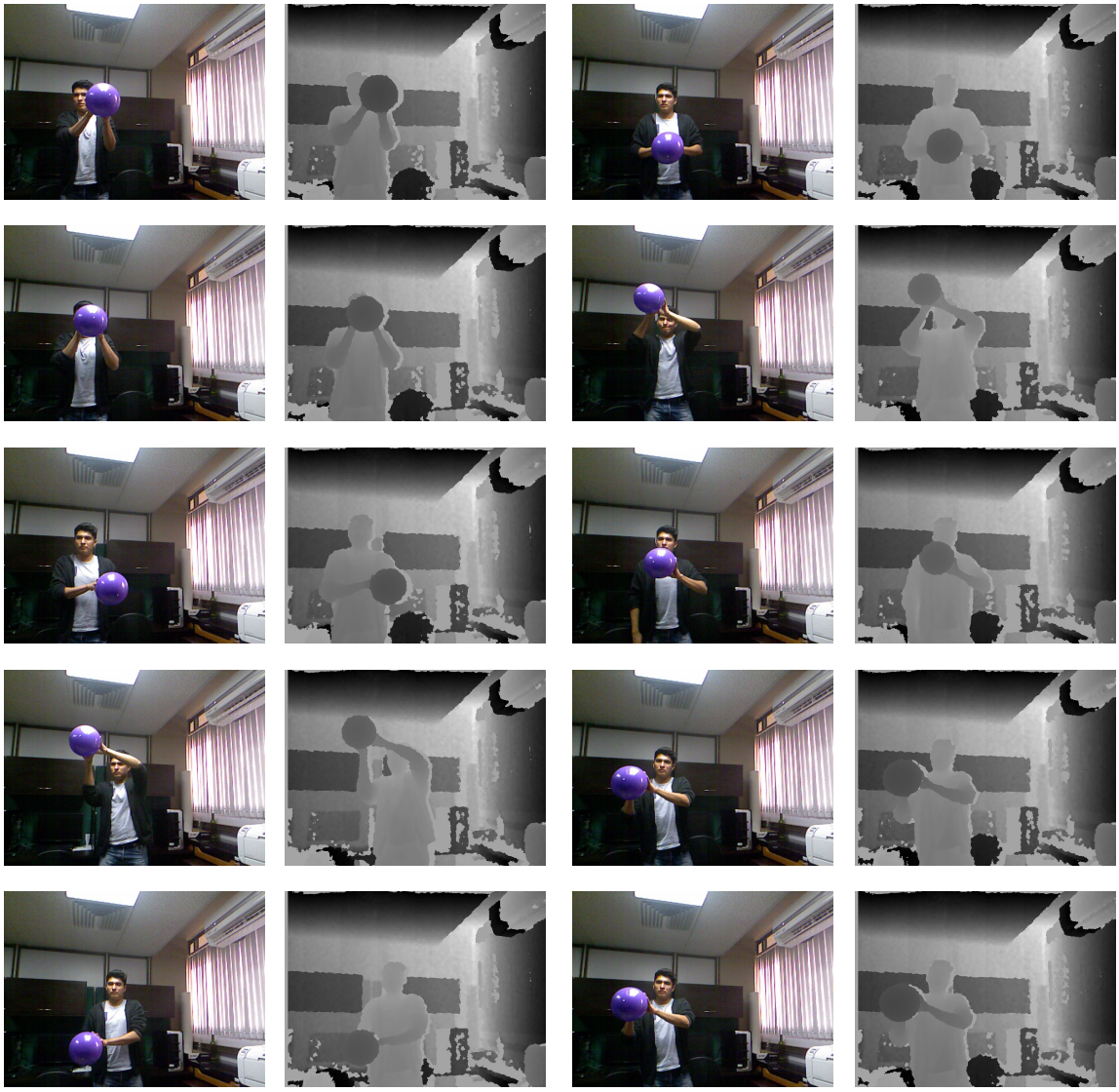


Figura 3.11: Captura de una esfera (pelota) en los espacios RGB y de profundidad

esferas son encontradas en ambos espacios, utilizando el método de ajuste de elipses por RANSAC, dando como resultado un radio r y un centro O. Con estos valores se puede representar una esfera en el espacio de profundidad, mediante una matriz [Hartley and Zisserman, 2003] según la Ecuación 3.7:

$$S = \begin{bmatrix} I_3 & -O \\ -O^T & O^T O - r^2 \end{bmatrix} \tag{3.7}$$

La esfera S proyectará una esfera E en el espacio RGB, siguiendo la Ecuación 3.8:

$$E^* = PS^*P^T \tag{3.8}$$

Siendo P la matriz descrita en la Ecuación 3.6 y S^* la matriz adjunta correspondiente a la Ecuación 3.7.

Conociendo la representación de la misma esfera en ambas imágenes, se puede utilizar un procedimiento de minimización no lineal para obtener la máxima verosimilitud de dos funciones propuestas por Staranowiks: L_1 basada en los centros de las esferas y L_2 basada en el contorno de la esfera según se observa en las Ecuaciones 3.9 y 3.10.

$$L_1 = \frac{1}{2N} \sum_{i=1}^{N} \epsilon_i^T \phi^{-1} \epsilon_i \tag{3.9}$$

$$L_1 = \frac{1}{2N} \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \left\| E_i^* - P S^* P^T \right\| \tag{3.10}$$

Donde el parámetro ϵ es el que da sentido a L_1 mediante la diferencia de los centros:

$$\epsilon_i = \tilde{O}_i - \frac{1}{\lambda_i} P O_i$$

Siendo λ_i un parámetro de escala, mientras ϕ y σ son la covarianza y la varianza del ruido medido, respectivamente, y deben ser escogidas para ser una función cuadrática de la distancia entre las esferas.

Finalmente, se puede minimizar la función objetivo de la Ecuación 3.11, que resulta de una combinación lineal de las Ecuaciones 3.9 y 3.10 ponderada por los valores ρ_1 y ρ_2:

$$\rho_1 L_1 + \rho_2 L_2 \tag{3.11}$$

El resultado de minimizar la Ecuación 3.11, dará como resultado las interrogantes de la matriz de transformación de la Ecuación 3.6, con la que podemos pasar de un espacio a otro.

Capítulo 4

Extracción automática de *sketches*

Como se definió en el Capítulo 1, un *sketch* es la abstracción monocromática de un objeto, tal que se conserven únicamente sus características principales de forma y se descarten detalles, como texturas, iluminación, sombras y demás especificaciones que intenten dar realismo al objeto, de forma gráfica. Esta definición dice que, en general, no es posible obtener el *sketch* de un objeto únicamente mediante su contorno (en realidad es posible, pero en condiciones bien controladas y con objetos preparados para ello). Por consiguiente, el proceso de extracción automática de *sketches* puede apoyarse de la detección de bordes o equinas, pero esto conlleva más etapas de procesamiento.

Como se deduce de la metodología propuesta en la Sección 1.4, la extracción de un *sketch* conlleva la extracción del fondo de las imágenes RGB y de profundidad (Sección 4.1). Esto se debe a que nuestra propuesta está pensada como una ayuda para métodos de navegación y visión robótica, por lo que, al igual que la visión humana, debe centrarse en un objeto en particular y despreciar el fondo innecesario. La extracción del fondo requiere una normalización del histograma de la imagen de profundidad (Sección 4.2) para mapear a un nuevo espacio que permita separar el fondo tanto de la imagen RGB como de la imagen de profundidad. Una vez tratadas dichas imágenes, es posible extraer los contornos de ambas imágenes con una técnica que permita una extracción rápida, como la presentada en la Sección 4.3. Sin embargo, como se verá en la Sección 4.4, no todos los contornos son necesarios, por lo que el procesamiento puede ser inteligente, i.e., no es necesario recorrer toda la imagen.

Para el resto de este capítulo, se utiliza como referencia la Figura 4.1, donde se puede observar una imagen RGB-D, separada en sus capas RGB y de profundidad. Esta separación ayuda a visualizar todo el proceso, el cual será descrito en las siguientes secciones.

a) b)

Figura 4.1: En a) se puede observar la imagen RGB, la cual ha sido escalada por motivos de presentación, pero el lector debe considerar que esta imagen es varias veces mayor que la de profundidad y que posee una relación de aspecto distinta. En b) se muestra la imagen de profundidad en bruto, aún sin haber sido tratada con la técnica de *inpainting* descrita en la Sección 3.2

4.1 Extracción del fondo

La extracción del fondo responde a nuestra intención de dejar claro desde el inicio con qué objeto se desea trabajar. Como es posible imaginar, este objeto se encuentra disponible sólo en una porción de la imagen en vez de estar en toda ella, lo que nos hace deducir que trabajar con la imagen completa, en todo el proceso descrito por el resto de esta tesis, sea inútil e innecesario. Basta con delimitar la región de nuestro interés, la cual será definida como la región central de la imagen, imitando el proceso de visión humana. Este no es un proceso restrictivo, ya que con la misma técnica, que se describe en esta sección, se pueden tomar diferentes porciones de interés.

Para una imagen RGB-D, existen tres posibles caminos a elegir cuando se trata con técnicas de extracción de fondos: utilizar la imagen RGB y procesar el fondo bajo las técnicas tradicionales, buscar un método que trabaje adecuadamente sobre el espacio de profundidad o hacer uso de las características de ambos espacios para obtener un algoritmo híbrido.

El estado del arte para la extracción de fondos es sumamente amplio. Al ser una técnica básica para el procesamiento de imágenes, las propuestas son abundantes, desde las aproximaciones que consideran fondos estáticos [Rodriguez-Gomez et al., 2015], hasta las técnicas más avanzadas que toman en consideración fondos dinámicos, texturizados o con gradientes de color. Ejemplos de estas últimas son las técnicas MOG (*Mixture of Gaussians*) [Hernández-Vela et al., 2012], las reglas Bayesianas de decisión [Li et al., 2003], los modelos basados en *Codebooks* [Fernandez-Sanchez et al., 2013], los estimadores de densidad del Kernel [Lee and Park, 2012], los algoritmos de *clustering* como el caso de K-means [Ryoo and Aggarwal, 2009] y, por supuesto, las técnicas de análisis de componentes, ya sea de componentes principales [Bravo et al., 2010] o de componentes independientes [Jiménez-Hernández, 2010]. Lo cierto es que, al traba-

jar sobre el espacio RGB, todas estas técnicas, aunque eficientes en buenas condiciones, se ven influenciadas por los factores descritos por el problema de la invariancia: sombras, iluminación, brillo y contraste, sólo por mencionar algunos; esto conduce nuestro pensamiento a la misma línea que hemos manejado hasta el momento, que es apoyarnos en los datos de profundidad.

En los últimos años se han desarrollado paradigmas híbridos que combinan características de los espacios de profundidad y RGB, dando como resultado una extracción robusta, que es fuerte ante el problema de la invariancia [Cristani et al., 2010]. Generalmente son utilizados como una mezcla de la aplicación en ambos espacios de un algoritmo diseñado para el espacio RGB, ayudándose de las disparidades de ambos espacios [Crabb et al., 2008] [Zhu et al., 2009a]. Si bien, en la práctica, tienen buenos resultados, debemos recordar que el espacio de profundidad representa coordenadas espaciales en vez colores (aunque se represente en escala de grises). En consecuencia, la aplicación de una técnica diseñada para RGB podría tener resultados inesperados. Por otro lado, esta combinación de espacios nos advierte que, al menos para la extracción del fondo, es necesario operar por completo en ambas imágenes, utilizando operaciones que pueden resultar costosas computacionalmente, por lo que este paradigma puede ser una opción, pero es necesario tener en cuenta estos dos factores.

Como tercera alternativa se tienen las técnicas basadas únicamente en la imagen de profundidad, las cuales pueden resultar adecuadas si se considera la siguiente definición de fondo (*background*): es la parte de la escena menos significativa y que generalmente se encuentra más alejada de los objetos de interés [Bouwmans et al., 2008]. Bajo esta definición, es comprensible que la información de profundidad es importante. Muchos autores se han percatado de que, en una imagen RGB-D, resulta más provechoso y menos costoso computacionalmente trabajar únicamente sobre imágenes de profundidad en vez de la mezcla de ambas, ya que, así como las ventajas de ambas se pueden adicionar, también lo harán sus desventajas y la imagen RGB es especialmente propensa al problema de la invariancia La mayoría de los trabajos en espacios de profundidad, para la extracción del fondo, mapean técnicas diseñadas para el espacio RGB. [Schiller and Koch, 2011] utiliza MOG para la substracción del fondo, mientras [Elhabian et al., 2008] muestra la adaptación de varias técnicas al espacio de profundidad, advirtiendo que las técnicas que trabajan sólo en espacio de profundidad tienden a dejar huecos y no reconocerán objetos delante del corte de proximidad o detrás del corte de lejanía, por lo que deben ser usadas con cuidado.

Por motivos del costo computacional, en esta tesis se usa una técnica basada únicamente en el espacio de profundidad, ya que es posible sortear las problemáticas que trae consigo, pues en nuestro caso sólo se analiza un objeto a la vez. En consecuencia, sólo es discernir entre el fondo y el objeto de análisis, de manera que los huecos en el fondo no serían un problema real, todo lo que está fuera del objeto central es considerado como parte del fondo. Además, por la amplitud de rango de nuestro sensor, los cortes de proximidad y lejanía tampoco son un problema. Por el contrario, afrontar los problemas de invariancia de las imágenes RGB o el tiempo de procesamiento

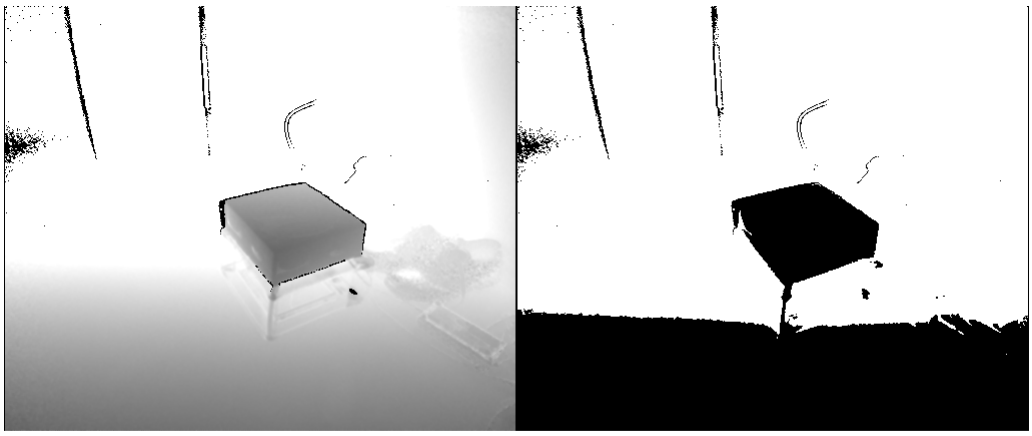


Figura 4.2: En a) se puede apreciar la imagen original, como se presenta en la Figura 4.1, mientras en b) se observa la aplicación de la técnica de Otsu. Como se aprecia, todos los píxeles cercanos a la cámara pasan a ser parte del objeto, pues tienen valores de proximidad (niveles de gris) mayores

necesario por los algoritmos híbridos sería mucho más complicado.

Como se ha reiterado, no es conveniente utilizar solo una técnica diseñada para espacios RGB, ya que como se observa en la Figura 4.2, la aplicación de una técnica sencilla como el método de Otsu [Otsu, 1979], a la imagen que se muestra en la parte b de la Figura 4.1, falla al separar el fondo porque el sensor percibe partes del fondo que están incluso a menor distancia que el objeto de interés. Por esta razón, las tonalidades de gris del fondo pueden ser superiores a las del objeto de interés.

Para solucionar esta problemática, se ha desarrollado una técnica que se basa en las apariciones consecutivas de un color o de su crecimiento lineal (contribución de esta tesis). El algoritmo recorre una imagen línea por línea y, en cada iteración, calcula todos los valles y crestas de la gráfica que se forman, tomando ventaja del ruido propio del sensor que provoca constantes variaciones de nivel; de las crestas sólo se conserva el máximo, mientras que de los valles se eliminan los valores máximos y mínimos; después, se calcula la media, la cual puede denotarse gráficamente como una línea de corte para la umbralización, como se aprecia en la Figura 4.3. Ahora bien, si la diferencia entre la media de los valles y el máximo valor de las crestas es menor que un valor $\alpha \in [3, 15]$, entonces se considera que toda la línea es parte del fondo, pues la variación de profundidades no es suficiente como para considerar que existe un objeto importante. Por otro lado, si la diferencia es mayor que $alfa$ entonces umbralizamos la línea, utilizando como umbral la media de los valles. Algunos huecos pueden quedar debido a la umbralización línea por línea, pero una sencilla operación de cierre morfológico resuelve el problema. El resultado es una máscara binaria que permite aislar los principales objetos, como se observa en la Figura 4.4 que, en comparación con la Figura 4.2, presenta una mejoría notable. Este sencillo algoritmo procesa píxel

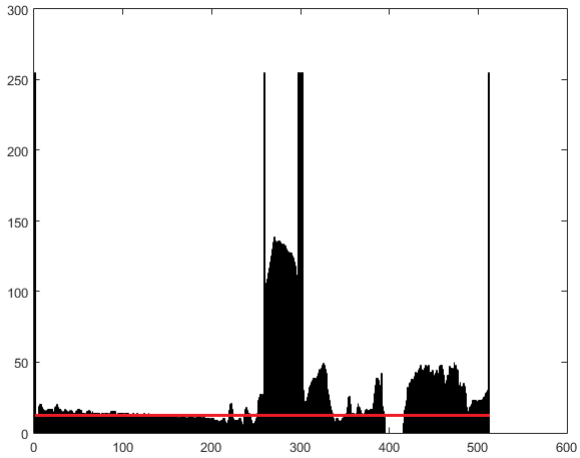


Figura 4.3: Niveles de gris o de profundidad en una linea de la imagen de profundidad. No se trata de un histograma. Se forman crestas y valles por el ruido y se aprecia que la curva Gausiana más alta representa la aparición del objeto de interés en esta linea, mientras que la linea roja muestra el nivel de fondo calculado.

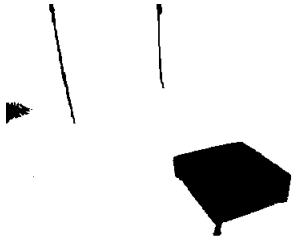


Figura 4.4: Mascara de separación del fondo resultante de aplicar el algoritmo 1

por píxel sin ciclos, lo que da una complejidad lineal $O(n)$.

Una vez diseñada la máscara para extraer el objeto de interés, se procede a aplicarla sobre la imagen. Sin embargo, ya que sólo estamos interesados en uno de los objetos, entonces se utilizará el objeto más cercano al centro de la imagen. El Algoritmo 1 muestra el proceso completo de la extracción del fondo.

4.2 Normalización del histograma de profundidad

Aplicada la máscara obtenida en la Sección 4.1 y extrayendo de ella únicamente el objeto central, ya tenemos nuestro objeto aislado, como se muestra en la Figura 4.5. Dependiendo de sus proporciones, este objeto sólo se encontrará definido en una parte

```
Input: Image I
[x,y]=size(I);
Mascara=empty();
for i=0 to x do
    posibleValle=false;
    posibleCresta=false;
    crestas=empty();
    valles=empty();
    for j=0 to y do
        if I(i,j)¡I(i,j-1) then
            if posibleCresta==true then
                crestas.add(I,j-1);
                posibleCresta=false;
            else
                posibleValle=true;
        else
            if posibleValle==true then
                valles.add(I,j-1);
                posibleValle=false;
            else
                posibleCresta=true;
    end
    level=mean(valles);
    if ‖max(crestas) − level‖ < α then
        Mascara(:,j)=fillLine(0);
    else
        Mascara(:,j)=binarize(I(:,j),level);
end
Mascara=cerradura(Mascara);
return Mascara;
```

Algoritmo 1: Extracción del fondo

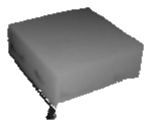


Figura 4.5: Después de aplicar la máscara mostrada en la Figura 4.4, se toma únicamente el objeto más cercano al centro, aislando así el objeto de interés

muy pequeña del espectro de profundidad, mientras que una gran sección del espectro estaría desocupada, como se observa en la Figura 4.6. Esto sería perjudicial para el algoritmo definido en la Sección 4.3, pues el objeto aislado representaría un bajo contraste en la imagen que no puede ser detectado correctamente como un contorno.

Para prevenir que todo el objeto tenga una misma tonalidad de gris (pues los píxeles del objeto aislado tienen valores de profundidad similares) o que el algoritmo de extracción de contornos se quede estancado en un rango muy pequeño. Dependiendo del sensor de profundidad que se tenga, es posible aplicar dos técnicas. El primer método es genérico para cualquier sensor y consiste simplemente en expandir el histograma en la región de interés, mediante una multiplicación escalar y una traslación, permitiendo crear valores intermedios que serán mejor diferenciables, al aplicar un algoritmo de extracción de contornos. Para aplicar esta técnica, es necesario encontrar los valores de corte de la curva del histograma, proceso idéntico al que es aplicado por el algoritmo de [Otsu, 1979]. Conocidos los valores del corte, se mapea el rango en el que se encuentran hacia un espacio de 0 a 255 (este sería el rango ideal, pero en la práctica un rango de 20 a 235 funciona mucho mejor). Es necesario recordar que estos valores no están necesariamente centrados, por lo que una traslación de los valores será requerida.

El problema de esta primera aproximación es que se crean datos artificiales, interpolando su valor, lo cual es adecuado en figuras regulares y simples, pero en figuras más complicadas este método puede restarles calidad. Por ello, una solución más acertada suele tomar otra imagen de profundidad justo al terminar la extracción del fondo y calibrar el sensor de profundidad para trabajar en un rango más acotado. Esto es posible para sensores como Kinect, como es nuestro caso, pero resulta difícil para otros sensores. Para obtener los valores del rango deseado, usaremos el mismo algoritmo

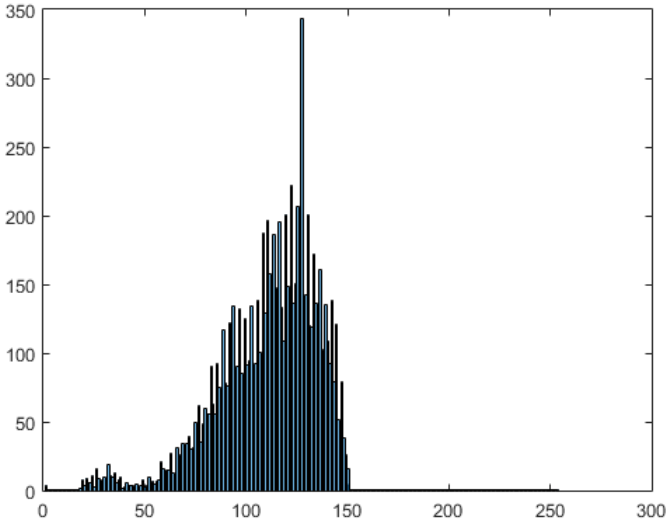


Figura 4.6: El objeto de interés solo ocupa una parte del espacio de profundidad definida entre los valores 50 y 150, aproximadamente.

de Otsu, igual que en el método anterior, pero esta vez lo usamos para configurar el sensor. El resultado es una imagen con información más útil para el proceso de extracción del contorno, como se observa en la Figura 4.7, en la que el objeto no ocupa más que el cuadrante central al hacer una división por tercios de la imagen. De hecho, esto es una reducción de un noveno en comparación con el tamaño original del objeto. Para una imagen RGB de Kinect, esto es una reducción de una imagen de 1280 x 720 píxeles a una de 427 x 240 píxeles, mucho más sencilla de procesar. Los objetos más grandes que se procesarán en el Capítulo 7, no ocupan más allá de un tercio del total del espacio de la imagen (1280 x 240 píxeles o 427 x 720 píxeles); esto simula la vista que tendría un robot de su espacio, mediante un sensor similar a Kinect. De esta manera, se observa que, en general, el número de píxeles a procesar se encuentra en el quinto orden de magnitud (10^5). El algoritmo de Otsu generalmente tiene una complejidad computacional alta pero, gracias a una modificación reportada en [Zhu et al., 2009b], es posible lograr una complejidad de $O(L^2)$ con $L^2 << N$ donde L es el número de niveles de gris y N es el número de puntos en la imagen de entrada.

4.3 Extracción de contornos y esquinas

En esta sección se explica la técnica de extracción de contorno utilizada, cuya utilidad recae en la creación del *sketch*. Los procesos anteriores en este capítulo, permitían centrarnos en el objeto de interés y resaltarlo. Aunque gracias a esta delimitación se puede trabajar con una cantidad de información mucho menor, es posible reducir aún

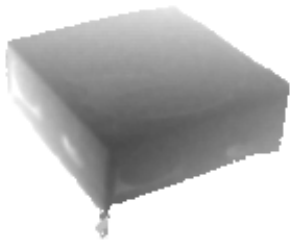


Figura 4.7: En comparación con la Figura 4.5, la nueva imagen tiene mejor contraste y sus bordes son mejor apreciables.

más la información que será procesada. Como se observa en la Figura 4.7, mucha de la información de profundidad no es relevante, el objeto en dicha figura fácilmente podría ser representado por sus contornos, como se observa en la Figura 4.7, el contenido entre los bordes es solo un degradado de grises, lo que se traduce en una cantidad de información muy pobre almacenada en un espacio muy amplio. En esta sección, se detallará el algoritmo utilizado y en la Sección 4.4 se hablará sobre cómo aplicar el contorno para crear el *sketch*, sin incluir detalles o efectos de iluminación.

Existe un estado del arte sumamente amplio en cuanto a la extracción de contornos, desde los clásicos métodos de primer orden como los operadores de Canny [Canny, 1986], Sobel [Duda, 1973] y Prewit [Prewitt, 1970], los operadores de segundo orden como los operadores de Marr-Hildreth [Marr and Hildreth, 1980] y Lindeberg [Lindeberg, 1996], hasta los métodos de congruencia de fase como la propuesta por [Arbelaez et al., 2011], los inspirados en Física [Asghari and Jalali, 2014] y por supuesto el los métodos de *subpíxel*. Entre esta multitud de técnicas, es fácil perder de vista que la idea es que el método elegido debe satisfacer el problema a resolver. En nuestro caso, es necesario priorizar la velocidad, por lo que hemos decidido desarrollar nuestro propio algoritmo de detección de bordes, el cual se basa en un algoritmo rápido y eficiente para la detección de esquinas, que fue desarrollado por [Rosten et al., 2010]. Como se verá, es posible hacer la detección de bordes y de esquinas en un solo paso, lo que nos motiva a usar este algoritmo.

El algoritmo de Roster et al., conocido como *Fast Corner Detection* (FCD) es sumamente sencillo. Pensemos en el análisis de un píxel para saber si se trata de un píxel común, un píxel de contorno o un píxel de equina. Cualquiera de los métodos antes mencionados requeriría operar con una máscara sobre el píxel y después reescribir su valor. Este proceso se aplica a cada uno de los píxeles de la imagen, pero el algoritmo FCD puede anticipar, con cierta probabilidad, si se trata de una esquina o un contorno, dejando sin operar todos los píxeles de relleno. Considerando que el contorno y las esquinas representan una mínima parte de la imagen, este proceso representa un considerable ahorro de tiempo.

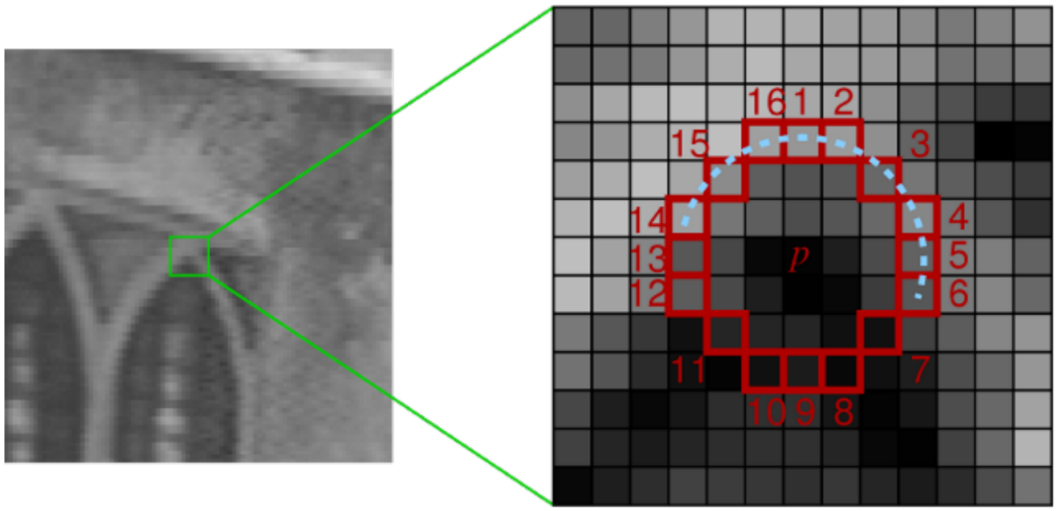


Figura 4.8: Un píxel reconocido como esquina y el círculo de Bresenham, junto con sus etiquetas de índice. Es posible observar que se cumple el postulado de Roster sobre los valores más claros o más oscuros que el píxel central.

Dado un píxel P, un circulo de Bresenham con centro en P y un radio r, así como un valor de umbralización t, el algoritmo FCD buscará las diferencias de intensidad entre el píxel P y los píxeles B_i que pertenecen al círculo de Bresenham. Típicamente se usa un valor $r = 3$ lo que nos da como resultado un círculo de Bresenham de 16 píxeles como se observa en la Figura 4.8. Se marcará un píxel como brillante si $v_i - v_p < t$ y como oscuro si $v_i - v_p < -t$, donde v_i es el nivel de intensidad del píxel B_i y v_p es el nivel de intensidad del píxel P. El píxel P se considera esquina si, por lo menos, 12 píxeles consecutivos del círculo de Bresenham son oscuros o brillantes. Como es de imaginarse, esto corresponde a un estudio visual de la imagen más que a un comportamiento matemático.

La parte más interesante de esta técnica es que no se tienen que evaluar los 16 píxeles del circulo de Bresenham, basta con evaluar los píxeles 1, 5, 9 y 13, según la Figura 4.8, i.e., los que corresponden a las posiciones del circulo cada 90 grados. Esto responde a la observación de que si y solo si tres píxeles consecutivos de esta lista de cuatro píxeles son oscuros o brillantes, se puede cumplir la condición de que 12 píxeles consecutivos lo sean. Si esta condición no se cumple, será imposible que existan 12 píxeles consecutivos oscuros o brillantes. Cuando la condición es cumplida, entonces se analizan todos los píxeles del círculo de Bresenham para verificar que existan 12 consecutivos, que sean oscuros o brillantes, y entonces poder marcar el punto como esquina. Cabe destacar que una vez encontrada una esquina, los píxeles adyacentes deben ser descartados de inmediato como posibles esquinas, ya que con la lógica definida por el algoritmo, varios píxeles adyacentes podrían cumplir la condición de los 12 píxeles consecutivos.

Al basarse en ID3, el algoritmo de Rosten et al tiene una complejidad de $O(dmLog(m))$ [Witten et al., 2016] con $d = 2$, por lo que se desprecia; m es el número de esquinas

en la imagen. Al simplificar y multiplicar por el número de puntos en la imagen se tiene $O(n \cdot mLog(m))$.

Parte del trabajo de esta tesis ha sido hacer una adaptación del algoritmo FCD para poder detectar bordes. El ajuste es simple, sólo se deben modificar los parámetros con los que trabaja el algoritmo. Como se observa en la Figura 4.8, se establece el parámetro de 12 píxeles consecutivos, ya que se considera una esquina, como tal, si tiene un ángulo de 90 grados. Para nuestros fines, es suficiente con que tenga 11 píxeles para considerarlo una esquina, ya que habrá algunas más abiertas que otras en los objetos. Sin embargo, la prueba de los tres de cuatro píxeles de las cuatro principales direcciones se mantiene. El sentido común nos diría que, para considerar que un píxel es parte de un borde, por lo menos la mitad del círculo debe tener un valor oscuro o claro consecutivo. En la práctica, esto no es del todo cierto y muchas veces existen bordes más cerrados, por lo que se establecerá el valor en 7. No obstante, al hacer la prueba de los tres píxeles consecutivos, se observa que, al tener una longitud de 7 píxeles, solo es posible cumplir con dos píxeles consecutivos de la prueba de cuatro píxeles y la prueba pierde significancia, pues el hecho de que dos píxeles consecutivos sean claros u oscuros tiene una probabilidad del 50 %, lo que no da ninguna certeza. Para volver a tener un índice adecuado de certeza, se hará una variación en los puntos de verificación. Entonces, en lugar de hacerlo cada 90 grados, se hará cada 60 grados. Ahora bien, como 16 no es divisible de forma entera entre 6, entonces se toman las mejores aproximaciones, de forma que los nuevos píxeles de verificación son 1,3,6,9,12 y 15. De esta manera, cuando se tiene que tres de estos son oscuros o brillantes y además son consecutivos, se obtiene una buena probabilidad de que el punto analizado sea un borde. Entonces, se procede a analizar todos los píxeles de esa área en el círculo de Bresenham.

Al adecuar nuestra nueva propuesta para detectar bordes, se puede ver que para un arco con longitud de 11 píxeles (i.e., cuando se quiere detectar una esquina) se cumple que, al menos, cuatro píxeles consecutivos de los píxeles de verificación son oscuros o brillantes. Si es así, es posible verificar la existencia de un arco consecutivo de longitud 11 de píxeles oscuros o brillantes. Si se cumple la condición, entonces se habrá encontrado una esquina. Es necesario recordar que, al igual que en la propuesta original, se deben descartar los píxeles adyacentes que tiendan a engrosar el contorno.

Para imágenes de naturaleza, este algoritmo puede no marcar todas las esquinas o todos los bordes. Sin embargo, en imágenes de objetos hechos por el ser humano, los resultados son sobresalientes, pues permite encontrar todas las esquinas y bordes en un tiempo menor, al descartar la mayoría de los puntos por anticipado, gracias a los píxeles de verificación.

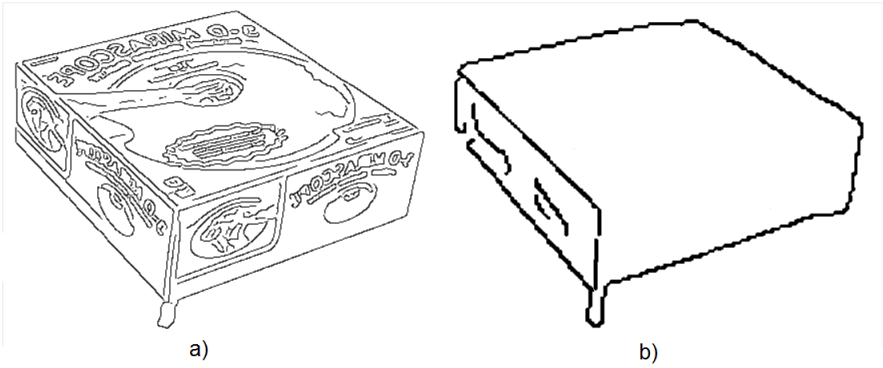


Figura 4.9: En a) se aprecian la imagen de contornos, a partir de la imagen RGB, así como una gran cantidad de contornos debidos a las texturas e iluminación. En b) se observa la imagen de contorno, a partir de la imagen de profundidad; la imagen cuenta con mucho ruido, producto del sensor. El contorno más grueso en b) es producto únicamente del escalamiento aplicado por motivos de presentación.

4.4 Emparejamiento de contornos y esquinas

Hasta el momento sólo hemos trabajado con la capa de profundidad, pero en este momento la capa RGB aportará información importante. Como podemos observar en la Figura 4.7, la imagen resultante contiene mucho ruido producto del sensor, así como de la dimensión de la imagen. Es por ello que obtener únicamente el contorno de esta imagen puede no ser conveniente, problema que se acrecienta a medida que el objeto es más complejo.

Así como se aísla el objeto en la capa de profundidad, también es necesario aislarlo en la capa RGB para no tener que trabajar con toda la imagen y, por lo tanto, se complique el proceso. Existen muchas técnicas completas que permitirían realizar este proceso, sin embargo, al trabajar con la capa RGB, nos enfrentamos nuevamente a la influencia de texturas, iluminación, coloración, entre muchos otros factores. Afortunadamente, no hay necesidad de repetir el proceso y enfrentar estas adversidades, pues una vez conocida la máscara de extracción del fondo (calculada en la Sección 4.1) sólo tenemos que aplicarle la matriz de transformación de la Sección 3.3. Este proceso permitirá tener una máscara de extracción del fondo en el espacio RGB y, de esta manera, aislar el objeto en el espacio RGB, al igual que en el espacio de profundidad.

Al aplicar el algoritmo descrito en la Sección 4.3 a los objetos aislados en las imágenes RGB y de profundidad, se obtienen los resultados que observamos en la Figura 4.9. Como se puede apreciar, la imagen de profundidad tiene problemas con el ruido, mientras la capa RGB tiene problemas con las texturas y la iluminación. También se puede observar que ninguna de las dos imágenes obtenidas de la extracción de contornos es un *sketch* por sí mismo.

La comparación entre los contornos del espacio de profundidad y los contornos del espacio RGB sí permitirá crear una mejor aproximación para obtener el *sketch* de

un objeto. Para ello, se tiene que recorrer la imagen del contorno en RGB y utilizar la matriz de transformación de la Sección 3.3. La idea básica del algoritmo de emparejamiento es la siguiente: se recorre la imagen de RGB, de manera que cuando se encuentre un píxel P_{RGB} de contorno (que también puede ser un píxel esquina), entonces se aplicará la matriz de transformación para saber a qué punto de la imagen de profundidad pertenece. Es decir, se obtendrá un punto P_{Depth}, si el píxel P_{Depth} es un píxel de contorno y entonces se deja también el píxel P_{RGB} tal como está. Si por el contrario, el píxel P_{Depth} no está marcado como un píxel de contorno, entonces el píxel P_{RGB} se marcará como vacío.

Teóricamente, el algoritmo de emparejamiento es correcto. Sin embargo, debido a la lógica entera, no siempre se obtendrá el píxel P_{Depth} que corresponda exactamente con el contorno buscado. Por ello, es necesario aplicar una máscara de convolución de kxk con k impar y con todos sus elementos en 1. Si el resultado de la operación es mayor que cero, entonces se encontró un píxel de contorno alrededor de P_{Depth} y P_{RGB} también se debe considerar como un píxel de contorno. Por el contrario, si el resultado de la convolución es cero, entonces se marcará como vacío el píxel P_{RGB}. Si bien n puede crecer todo lo que el programador considere necesario, nuestras pruebas demuestran que los mejores resultados se obtienen con $k = 3$ y $k = 5$; valores superiores de n traen como resultado el cierre de grandes huecos (lo que a veces es una ventaja y a veces es perjudicial) y la inclusión de contornos de texturas cercanas a los bordes.

Es importante recordar que, en este punto del procesamiento, tanto la imagen de profundidad como la imagen RGB se han convertido en imágenes binarias y de una dimensión menor a la original, debido al procesamiento hasta ahora efectuado en ellas. Por lo tanto, el tiempo de procesamiento de esta etapa es sumamente rápido. Además, se debe considerar que la máscara de convolución se aplica únicamente cuando se encuentra un píxel de contorno y estos son los casos menos frecuentes en la imagen RGB.

Sea C una imagen del espacio RGB de tamaño $n \times m$, que presenta el objeto asilado y a la cual se le ha aplicado el algoritmo de contornos y esquinas de la Sección 4.3 y sea D la imagen de profundidad con el objeto aislado a la cual se le aplica el mismo algoritmo para contornos y esquinas. Entonces, el Algoritmo 2 presenta, en forma resumida, cómo aplicar el emparejamiento para generar un *sketch* apoyado en las capas RGB y de profundidad.

La aplicación del Algoritmo 2 sobre las imágenes C y D da como resultado las imágenes presentadas en la Figura 4.10. A la izquierda, se puede observar lo que podría ser considerado como el *sketch*. Si bien existen algunas aperturas entre líneas y un par de puntos que lograron pasar entre los filtros, estos son fácilmente descartables con operaciones morfológicas. Sin embargo, esto no será necesario, como se verá en el capítulo 4, pues como se aprecia en la parte derecha de la Figura 4.10, estas aperturas y puntos aislados provocan esquinas muy cercanas a las líneas de contorno, lo que en la segmentación de la imagen será interpretado como si fuera parte de la misma

```
Input: C, D, M=K[Rt], n
mask=ones(n,n);
for i=0 to n do
    for j=0 to m do
        if C(i,j) then
            [x, y] = M * [i, j]^T;
            a=convolucion(D,mask,x,y);
            if a > 0 then
                C(i,j)=1;
            else
                C(i,j)=0;
    end
end
return C;
```

Algoritmo 2: Emparejamiento de contornos

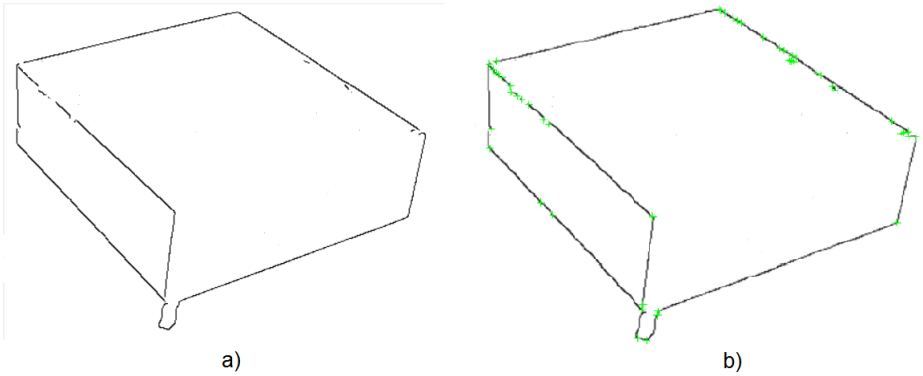


Figura 4.10: En a) se observa el resultado final del proceso para obtener el *sketch*; el resultado tiene buena definición y presenta poco ruido. En b) se han agregado las esquinas calculadas que serán útiles en el Capítulo 4

línea. Este algoritmo sólo recorre los puntos de contorno. Así, suponiendo que se han obtenido H puntos de contorno, la complejidad del método de emparejamiento es de $O(k^2H)$.

Capítulo 5

Segmentación convexa

La separación de un objeto en componentes convexos es uno de los problemas fundamentales a resolver durante esta tesis, ya que en el capítulo 6 se resolverá el ajuste de primitivas tridimensionales ($fitting$), las cuales son convexas, i.e., cuando existen concavidades en una figura compleja es necesario separarlas para obtener las primitivas tridimensionales más básicas que componen al objeto. Esta no es una actividad trivial, pues cuando una figura presenta huecos (y en un objeto hecho por el ser humano la existencia de huecos es común), la descomposición en componentes convexos se vuelve un problema $NP - Hard$ [Lingas, 1982].

Muchas aproximaciones han sido propuestas para la resolución del problema de descomposición convexa. De hecho, existen varias aproximaciones con tiempos de ejecución muy eficientes, siempre y cuando se pueda asumir que no existen huecos. Un ejemplo es el trabajo de [Chazelle, 1982] que tiene un tiempo de ejecución $O(nlogn)$, pero produce casi cinco veces más componentes convexos que el óptimo. Una medida con menos componentes convexos fue desarrollada por [Keil and Snoeyink, 2002] con un tiempo de ejecución $O(n+r^2min(r^2,n))$, donde r es el número de cortes necesarios para producir nuevas figuras convexas.

Cuando es necesario que el algoritmo de segmentación tenga en cuenta los huecos, el problema se complica, pues será necesario tomar en consideración los puntos de Steinger [Lien and Amato, 2006] para la correcta partición, conduciendo a un problema $NP - Hard$. Sin embargo, buenas aproximaciones han sido descritas, como la propuesta por [Chazelle and Dobkin, 1985], que tiene un tiempo de ejecución $O(n + r^3)$, el cual es aceptable cuando los huecos son figuras simples o existen pocos huecos, ya que si r se aproxima a $\Theta(n)$, el orden real del algoritmo se acerca a $O(n^3)$ como demuestra [Lien and Amato, 2006]. Aproximaciones más recientes, como las descritas en [Liu et al., 2010] [Lien and Amato, 2006] [Ren et al., 2013], tienen tiempos de ejecución $O(n^2)$, sin embargo, la operación de partición sigue siendo un problema computacionalmente costoso.

El objetivo de este capítulo es hacer uso de las características geométricas de los objetos hechos por el ser humano y presentar un algoritmo de segmentación que

tenga un tiempo de ejecución competitivo, en general, pero eficiente para el propósito de esta tesis. En las siguientes secciones construiremos el proceso de segmentación a partir de *sketches* con sus puntos de esquinas y bordes, obtenidos como resultado del proceso descrito en el Capítulo 4. Así, en la Sección 5.1 se construye la entidad matemática que permite conocer la relación entre las diferentes esquinas de un *sketch*. En la Sección 5.2 se presenta un algoritmo de cubierta convexa que, en general, se comporta mejor que los algoritmos relacionados y que además es incremental, lo que permite definir fácilmente puntos de inflexión. Utilizando los algoritmos y conceptos presentados en las Secciones 5.1 y 5.2, se propone un algoritmo sencillo de complejidad $O(nlogn)$ para la segmentación convexa, el cual es descrito en la Sección 5.3.

5.1 Grafo de puntos de interés

El resultado obtenido del proceso descrito en el Capítulo 4 es una imagen, cuyos bordes y esquinas son conocidos. Esto da pie a una nueva interpretación más simple de la imagen: un grafo. El procesamiento de imágenes como grafos para minimizar el espacio en disco es una práctica común, al igual que el uso de las esquinas para hacer la identificación de una imagen, como lo hacen los descriptores de imágenes. Esta representación es muy útil matemáticamente, pues es posible avanzar y retroceder rápidamente en la estructura de datos, marcar los nodos visitados y conocer sus vecinos, todo ello sin operar constantemente sobre la imagen.

La propuesta del uso de grafos está basada en el trabajo de [Guo et al., 2007] para hacer de cada esquina del *sketch* un vértice del grafo y crear relaciones lineales con sus vecinos. Si bien Guo et al también toman en cuenta la textura, esta queda fuera de los objetivos de nuestro trabajo, por lo que se hará omisión de esta parte de la representación. Así, sólo trabajaremos con las esquinas y los bordes para determinar pertenencia.

La representación ideal de un *sketch* vendría dada por un grafo $G = (V, E)$ que se forma por $V = v_1, v_2, \ldots, v_n$, donde cada v_i representa una y solo una de las esquinas del *sketch*, teniendo entonces que n es el número de esquinas de dicho *sketch*. Las aristas $E \subseteq \{e \in \mathcal{P}(A) : |e| = 2\}$ pertenecen a un conjunto potencia $\mathcal{P}(A)$ que ya se encuentra definido gracias al *sketch*, pues cada arista representa la comunicación entre dos esquinas, i.e., una arista e entre dos vértices v_i y v_j existe si y solo si, hay un borde en el *sketch* que es capaz de comunicar las esquinas que representan los vértices v_i y v_j. Cuando se introduzca la existencia de huecos en el *sketch*, esta definición deberá ser completada, pero por el momento es suficiente para comenzar la agrupación de esquinas, que es un paso previo e importante para la construcción del grafo.

Para formar el grafo debemos recorrer todas las esquinas del *sketch*. Para ello, se hace uso de los bordes del *sketch* como si fueran las aristas de un grafo, pero completando la información faltante (huecos que pueden formarse en medio de un borde) y eliminando

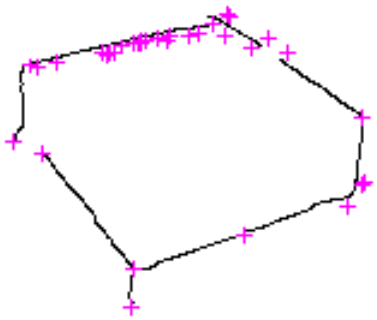


Figura 5.1: En este ejemplo se ha detectado una gran cantidad de esquinas debido al ruido, sin embargo, es obvio que muchas de ellas pueden agruparse como una sola.

la información duplicada (zonas donde el ruido puede provocar múltiples esquinas). Este proceso se ve mejor esquematizado en la Figura 5.1. En realidad, es posible llevar a cabo la eliminación de duplicados y cerrar los huecos en una sola operación de agrupación. Para ello se establece un parámetro τ tal cumple con la Ecuación 5.1:

$$\tau = \sqrt{\frac{r^2}{2}} \tag{5.1}$$

Donde r es el radio en el que consideramos que dos esquinas son la misma. Como se está tratando con un espacio discreto (los pixeles del *sketch*) se busca que τ sea entero, mientras que r puede ser real. Esta relación puede verse como una distancia euclidiana para su cálculo, como se observa en la Figura 5.2, pero en la práctica es mejor aplicarla como una distancia de Chebyshev, tal que $Chebyshev(v_i, v_j) \leq \tau$, aunque esto reduzca nuestra área de búsqueda a sólo el cuadrado circunscrito dentro del círculo de radio r.

La idea de la agrupación es simple: todas las esquinas son consideradas como vértices $v_i \in V$ y, en lugar de buscar recursivamente todos los puntos que, dado un vértice v_i, se encuentran a una distancia menor a r, se ordenan lexicográficamente todos los vértices por su coordenada Y. Este proceso toma un tiempo $O(nlogn)$ con un algoritmo de ordenamiento convencional (en nuestro caso hemos usado una versión iterativa de *QuickSort*) dándo como resultado una lista de vértices V_s. El siguiente paso es recorrer V_s determinando, por cada vértice v_i, la distancia de Chevyshev respecto a sus pares cercanos. Una forma óptima de hacer esto es definir una ventana $W = v_a, \ldots, v_i, \ldots, v_z$, donde v_a es el punto más alejado hacia abajo de v_i, tal que $v_{i_y} - v_{a_y} \leq \tau$ y v_z es el punto más alejado hacia arriba de v_i, tal que $v_{z_y} - v_{i_y} \leq \tau$. Hemos usado a propósito el verbo $definir$, ya que no es necesario hacer una copia de los elementos de W, sino que basta con conocer los índices de v_a y v_z para operar sobre W. Por cada elemento de W, es necesario verificar su coordenada X, de tal

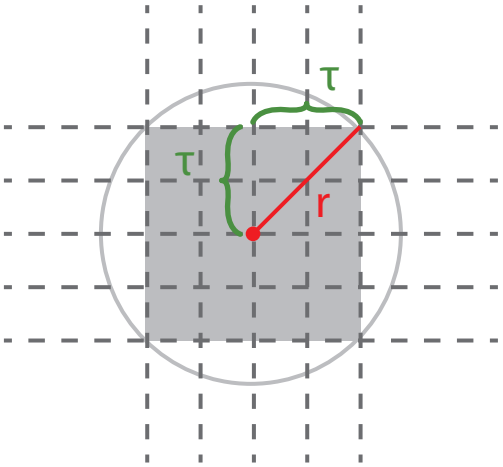


Figura 5.2: La búsqueda de pixeles usando la distancia de Chevyshev, dado un parámetro τ, resulta más eficiente computacionalmente que usar una distancia euclidiana con radio de búsqueda r, sin embargo, acorta la búsqueda a sólo el cuadrado circunscrito.

forma que se cumpla la Ecuación 5.2:

$$|v_{i_x} - v_{i+1_x}| \leq \tau \tag{5.2}$$

Todos los vértices en W, tales que cumplan la Ecuación 5.2, son esquinas tan cercanas entre sí que deben ser consideradas como la misma esquina; r es definida convenientemente según la experimentación; para nuestros experimentos $r < 15$, lo que da como resultado $0 < \tau \leq 10$, según la discretización de la Ecuación 5.1.

Cada vértice debe contener un campo adicional al de las coordenadas de la esquina que representa. Denominamos a este campo *ptr*, pues es un apuntador al vértice al cual queda ligado el actual vértice por considerarse que es la misma esquina. Un problema a afrontar es que un vértice ya se encuentre agrupado a otro o que se tengan dos o más grupos de vértices que se agruparán. Para resolver esta situación, se puede usar una lista auxiliar l a la que se agregan todos los apuntadores encontrados en el recorrido desde v_a hasta v_z. Al final, a todos los vértices en la lista se les asigna una sola referencia, la cual puede ser al vértice más cercano al centro de todos ellos o al primer vértice de la lista, dado el espacio reducido.

Una manera sencilla de llevar a cabo la comparación de la Ecuación 5.2 en la ventana W es mediante tres índices a, i, z que representan a los vértices v_a, v_i, v_z respectivamente; estos se inicializan como $a = i - 1$ y $z = i + 1$ y avanzan hasta alcanzar los límites marcados por τ. El proceso completo para la agrupación está resumido en el Algoritmo 3.

Ahora que se conocen los vértices y sus agrupaciones, es necesario construir sus relaciones, i.e., el conjunto de aristas E. Al igual que en el trabajo de [Guo et al., 2007],

```
Input: Vertex list V
Vs=lexicographicSort(V);
a=0;
z=2;
for i=1 to size(Vs)-1 do
    while Vs(i).y − Vs(a).y ≤ τ do
        a–;
    end
    a++;
    while Vs(z).y − Vs(i).y ≤ τ do
        z++;
    end
    z–;
    List l;
    for j=a to z do
        if |Vs(i).x − Vs(j).x| ≤ τ then
            if Vs(j).ptr!=NULL then
                l.add(Vs(j).ptr);
            else
                l.add(Vs(j));
    end
    Assign(l, l(0).ptr);
end
```

Algoritmo 3: Agrupación de vértices

usamos un proceso simple de inundación sobre el *sketch*, gracias a que el costo computacional no se eleva. Aunque es un algoritmo recursivo, al trabajar en una imagen en blanco y negro, que sólo tiene la silueta del objeto, se opera sobre muy pocos píxeles. Además, es sabido que los píxeles blancos no contienen información y que los píxeles negros son bordes o esquinas. Así, establecemos un nuevo tipo, al que se denomina "píxeles grises", que no son más que aquellos píxeles que ya han sido visitados durante la inundación. La única modificación del algoritmo de inundación tradicional es saber de qué vértice proviene un camino y, al encontrarse con otro vértice, enlazarlos mediante una arista.

Dado un vértice v_i asociado a una esquina con coordenadas (x, y), se busca si hay un camino en los 8 píxeles vecinos en la imagen C (la imagen de salida del proceso aplicado en el Capítulo 4), i.e., en los píxeles $C(x-1, y)$, $C(x-1, y-1)$, $C(x-1, y+1)$, $C(x+1, y)$, $C(x+1, y-1)$, $C(x+1, y+1)$, $C(x, y-1)$ y $C(x, y+1)$. Dicho de otra forma, se busca si alguno de estos píxeles es negro, se marca el píxel actual como gris y se avanza hacia los píxeles negros, enviando la información del último vértice visitado, i.e., la referencia a v_i, y repitiendo la misma prueba en sus correspondientes vecinos. De toparse con un píxel blanco o gris no se continúa con la recursividad, pero es necesario recordar que, para que este esquema funcione adecuadamente, las líneas deben ser tan delgadas como un píxel. De no ser así, se corre el riesgo de no alcanzar las esquinas y de terminar en un píxel gris, así como de crear recursividad excesiva en toda la línea, por lo que la operación morfológica de cerradura, aplicada a C en el Capítulo 4), es fundamental. Al avanzar a un píxel negro, se debe verificar si se trata de un vértice. Como ya se tiene un arreglo V_s de vértices lexicográficamente ordenados, la verificación se hace en un tiempo $O(logn)$. Además, cada píxel negro es visitado sólo una vez, aunque el algoritmo sea recursivo, por lo que el tiempo de ejecución es de $O(nlogn)$. Al encontrar que el píxel visitado v_j es un vértice, se genera una arista entre v_i y v_j. Sistemáticamente, esto se ve como agregar un elemento a la lista de direcciones asociadas de los vértices v_i y v_j. Cabe recordar que teóricamente un vértice puede estar enlazado a un número infinito de esquinas, pero en la práctica es poco común que un vértice se encuentre asociado a más de cuatro esquinas. El Algoritmo 4 muestra los pasos anteriormente descritos para crear todas las aristas del grafo.

El Algoritmo 4 inicia enviando el primer vértice en V_s y una referencia a sí mismo. Es de notar que el Algoritmo 4 toma en consideración las agrupaciones de los vértices y, si el apuntador de un vértice es diferente de vacío, la esquina de procedencia es asociada al vértice relacionado, no al vértice real.

Hasta el momento no se ha tomado en cuenta la existencia de huecos, pero estos juegan un papel importante, pues si bien muchos objetos hechos por el ser humano no tienen huecos, un grupo importante si los tiene. Es por ello que suponemos que es posible que existan huecos, por lo que puede existir un conjunto de vértices que no estén conectados por ninguna arista con el resto de los vértices. Entonces, no se cumple que n sea el número de esquinas del *sketch*, para $G = (V, E)$ con $V = v_1, v_2, \ldots, v_n$. Para

Input: reference Vertex v, Image C, Vextex list Vs, x, y
$C(x, y) = gris$;
index=SearchVertexIndex(Vs, x, y);
if *index!=NULL* **then**
  Vs(index).edge.add(v);
  v=Vs(index);
  if *Vs(index).ptr!=NULL* **then**
    Vs(index).ptr.edge.add(v);
    v=Vs(index).ptr;
  else
    Vs(index).edge.add(v);
    v=Vs(index);
if $C(x+1, y+1) == negro$ **then**
  aristas(v,C,Vs,$x+1$,$y+1$);
if $C(x+1, y) == negro$ **then**
  aristas(v,C,Vs,$x+1$,y);
if $C(x+1, y-1) == negro$ **then**
  aristas(v,C,Vs,$x+1$,$y-1$);
if $C(x-1, y+1) == negro$ **then**
  aristas(v,C,Vs,$x-1$,$y+1$);
if $C(x-1, y) == negro$ **then**
  aristas(v,C,Vs,$x-1$,y);
if $C(x-1, y-1) == negro$ **then**
  aristas(v,C,Vs,$x-1$,$y-1$);
if $C(x, y+1) == negro$ **then**
  aristas(v,C,Vs,x,$y+1$);
if $C(x, y-1) == negro$ **then**
  aristas(v,C,Vs,x,$y-1$);

Algoritmo 4: Creación de las aristas del grafo

representar el *sketch* es necesario un conjunto de grafos $H = G_1, G_2, \ldots, G_k$ donde $G_i = (V_i, E_i)$ y $|V_1| + |V_2| + ... + |V_k| = n$. Para obtener los k grafos que representan completamente el *sketch*, se tiene que aplicar iterativamente el Algoritmo 4, en el que todos los vértices marcados como grises durante el recorrido recursivo pasen a ser el conjunto V_i del grafo G_i. Si quedan vértices sin visitar, se toma el primero de ellos y se aplica el Algoritmo 4 y así sucesivamente hasta que no queden vértices sin visitar en el arreglo V_s.

El conjunto de grafos H es una abstracción matemática eficiente para que todos los procesos consecuentes en esta tesis puedan trabajar adecuadamente, pues es fácil identificar los huecos respecto al objeto principal (se espera que el área más amplia sea la del objeto principal y los demás grafos sean los huecos), ofrece tiempos lineales en su recorrido y presenta sólo información relevante.

5.2 Symmetric Hull

El problema de la cubierta convexa consiste en calcular, dado un conjunto D de puntos, el conjunto convexo más pequeño que contiene D [De Berg et al., 2008]. Los algoritmos de cubierta convexa se utilizan normalmente como una herramienta para resolver problemas más complicados en diversos campos, tales como Geometría Computacional (e.g., triangulación de Delaunay y diagramas de Voronoi), Reconocimiento de Patrones [Zhou et al., 2010], Procesamiento Digital de Imágenes [Messinger et al., 2010], Visión y Gráficos por Computadora [Ren et al., 2011], entre otros. Dichos algoritmos no son exclusivos del área de la Computación, pues tienen muchas aplicaciones científicas, como sistemas geográficos [Taillandier et al., 2010], análisis de información astronómica [J. Ostro et al., 2002], así como sistemas médicos y biológicos [Mosig et al., 2009] [Adanja et al., 2010].

Puesto que es un problema con repercusiones en diversos campos, existen múltiples maneras de resolverlo. De hecho, se han propuesto muchas soluciones basadas en algoritmos de clasificación, desde el enfoque de Chand y Kapur con complejidad $O(n^2)$ [Chand and Kapur, 1970] hasta otros enfoques clásicos con complejidad $O(nlogn)$, como Graham-Scan (uno de los más utilizados) [Graham, 1972], Monotone Chain [Andrew, 1979], Ultimate Planar Convex Hull [Kirkpatrick and Seidel, 1986], QuickHull (uno de los más rápidos) [Barber et al., 1996] y el algoritmo de Chan [Chan, 1996]. Además, dichos algoritmos podrían beneficiarse de la heurística de Akl-Toussaint [Akl and Toussaint, 1978] para la eliminación temprana de puntos innecesarios.

Como se mencionó anteriormente, los algoritmos de cubierta convexa representan una herramienta en otras áreas de investigación. Por esta razón, estos algoritmos podrían no ser un proceso final, sino que al mismo tiempo otros procesos se encontrarían en ejecución, como es nuestro caso, donde sólo queremos aplicarlo como parte del proceso de segmentación y, como el lector habrá notado, es una parte de toda la metodo-

logía necesaria para la reconstrucción y clasificación de objetos. En consecuencia, se han propuesto nuevos algoritmos, como el de Liu y Chen [Messinger et al., 2010], el cual aprovecha la geometría de cada cuadrante en un espacio 2D, o QuiGran [Sharif et al., 2011] que combina dos técnicas populares, Graham-Scan y QuickHull, con el fin de lograr mejores resultados que los obtenidos de estos dos algoritmos por separado. Por otro lado, la capacidad del hardware para procesar puntos continúa aumentando, por lo que la investigación en cubiertas convexas sigue activa, centrándose en la búsqueda de cómo calcular una gran cantidad de puntos en poco tiempo y de cómo ser invariante ante los pequeños cambios de posición introducidos por el ruido en los sensores.

SymmetricHull es la propuesta que hemos desarrollado para tratar grandes cantidades de puntos. Además, presenta la característica de poder ajustar sus parámetros para hacer frente a las variaciones de posición en los puntos obtenidos por los sensores, lo cual será de suma utilidad en el proceso descrito en la Sección 5.3. A continuación, se hace uso de la heurística de Akl-Toussaint para dividir el espacio en cuadrantes marcados por cuatro puntos extremos: la máxima ordenada (*top*), la mínima ordenada (*bottom*), la mínima abscisa ($leftmost$) y la máxima abscisa ($rightmost$). SymmetricHull sigue un enfoque geométrico, similar al del algoritmo de Liu y Chen [Liu and Chen, 2007], con la diferencia de que la eliminación de los puntos, al interior del paralelogramo descrito por los puntos extremos, es procesada al mismo tiempo que la creación de la cubierta convexa. Además, SymmetricHull aprovecha las características geométricas (de la pendiente con respecto a la convexidad) y simétricas (de reflexibilidad) de cada cuadrante para encontrar la cubierta convexa, realizando sólo una comparación de la pendiente entre dos puntos consecutivos, dentro un arreglo ordenado lexicográficamente por la coordenada y.

5.2.1 Contexto geométrico

Antes de aplicar un algoritmo de cubierta convexa, es habitual tratar de eliminar una importante cantidad de puntos, buscando formas geométricas al interior, como círculos [Liu et al., 2012] o paralelogramos [Liu and Chen, 2007]. Uno de los métodos más simples para suprimir puntos es el del máximo paralelogramo inscrito, el cual fue propuesto en la heurística de Akl-Toussaint [Akl and Toussaint, 1978]. Según este método, es necesario identificar los cuatro puntos extremos en las cuatro direcciones principales de un espacio 2D, como se enuncia a continuación.

Sea D el conjunto inicial de puntos y sean P_{Top}, P_{Bottom}, $P_{MostRight}$ y $P_{MostLeft}$ puntos que pertenecen a D y que representan los extremos en el eje de las ordenadas y de las abscisas, respectivamente. Estos puntos extremos dividen el espacio en cuatro cuadrantes Q_1, Q_2, Q_3 y Q_4, al mismo tiempo que forman las esquinas del máximo paralelogramo inscrito R, tal como se observa en la Figura 5.3.

Sea P un punto que pertenece al conjunto inicial D. Si P está dentro de R, entonces P no es parte de la cubierta convexa de D [De Berg et al., 2008], la cual se muestra

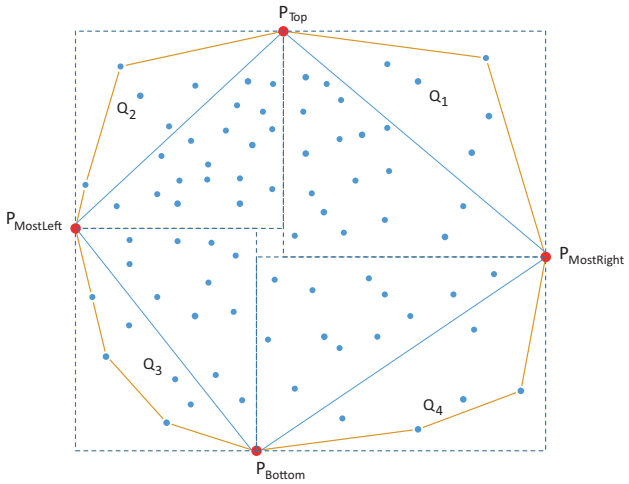


Figura 5.3: Conjunto inicial de puntos D, seccionado en cuadrantes por sus puntos extremos en rojo, que denotan los vértices del máximo paralelogramo inscrito R, el cual está representado por líneas continuas en azul, mientras en naranja se muestra la cubierta convexa de D

en la Figura 5.3.

Usualmente, una vez que el máximo paralelogramo inscrito de Akl-Toussaint ha sido encontrado, se eliminan los puntos dentro de R y posteriormente se aplica un algoritmo de cubierta convexa. En este caso, SymmetricHull junta ambas etapas, gracias a dos observaciones referentes a la geometría de los cuadrantes y a sus respectivas reglas, como se describe a continuación.

Sea D_S el resultado de ordenar lexicográficamente por la coordenada y, sea D el conjunto de puntos de entrada y sean $P(x, y)$, $P_A(x_a, y_a)$ y $P_B(x_b, y_b)$ puntos en D_S que pertenecen al mismo cuadrante, que son consecutivos, de forma que $y_a < y < y_b$, y que son parte de la cubierta convexa. Entonces, es posible hacer las dos observaciones siguientes:

Observación 1. Dado que los tres puntos, P, P_A y P_B, son candidatos a ser parte de la cubierta convexa, se cumple la condición de que P siempre está contenido en el rectángulo, cuya diagonal está definida por P_A y P_B, como se observa en la Figura 5.4. En realidad, P siempre se encuentra en uno de los triángulos o bien en la diagonal del rectángulo (en esta observación, basta con definir el rectángulo, ya que la posición del punto P será definida en la Observación 2).

La aseveración anterior se demuestra fácilmente ya que, por definición, P está entre P_A y P_B en el eje de las ordenadas (Y). En cuanto al eje de las abscisas (X), si P se encuentra a la izquierda del rectángulo, este punto no podrá ser parte de la cubierta convexa, ya que formará una concavidad en los cuadrantes Q_1 y Q_4; en el caso de los cuadrantes Q_2 y Q_3, el punto que formará la concavidad es P_B o P_A, dependiendo de si se analiza de arriba a abajo o viceversa. Del mismo modo, si P se encuentra a la derecha del rectángulo, entonces P_A o P_B formará una concavidad en los cuadrantes

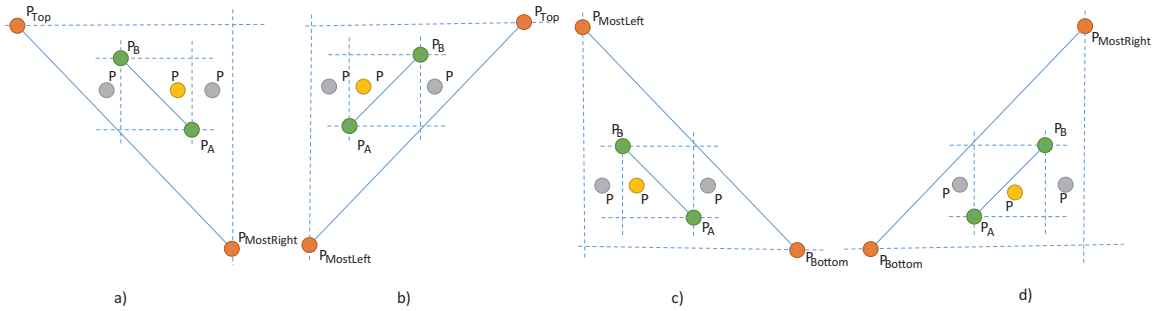


Figura 5.4: Rectángulos por cuadrante definidos por P_A y P_B que contienen al punto P, si este es parte de la cubierta convexa

Q_1 y Q_4 (dependiendo del sentido de análisis) mientras que, en los cuadrantes Q_2 y Q_3, el punto P formará dicha concavidad. De esta manera, las reglas 5.3 divididas por cuadrante se aplican para cualesquiera tres puntos, que presenten las características definidas para P_A, P y P_B:

$$\begin{aligned} Q_1 &: x_b <= x <= x_a \\ &\quad y_a <= y <= y_b \\ Q_2 &: x_a <= x <= x_b \\ &\quad y_a <= y <= y_b \\ Q_3 &: x_b <= x <= x_a \\ &\quad y_a <= y <= y_b \\ Q_4 &: x_a <= x <= x_b \\ &\quad y_a <= y <= y_b \end{aligned} \tag{5.3}$$

Observación 2. Suponga que el arreglo D_S se recorre de arriba hacia abajo para los cuadrantes Q_1 y Q_2, y de abajo hacia arriba para los cuadrantes Q_3 y Q_4. El comportamiento de la pendiente entre dos puntos será creciente para los cuadrantes Q_2 y Q_4, y será decreciente para los cuadrantes Q_1 y Q_3, al moverse de afuera hacia adentro en cada cuadrante, como se ilustra en la Figura 5.5.

Entonces, se puede definir s_a como la pendiente entre P_A y P, y s_b como la pendiente entre P y P_B para especificar, de esta forma, las Reglas 5.4 que describen el comportamiento de la pendiente al girar desde una posición horizontal hacia una posición vertical:

$$\begin{aligned} Q_1 &: s_b \geq s_a \\ Q_2 &: s_b \leq s_a \\ Q_3 &: s_b \leq s_a \\ Q_4 &: s_b \geq s_a \end{aligned} \tag{5.4}$$

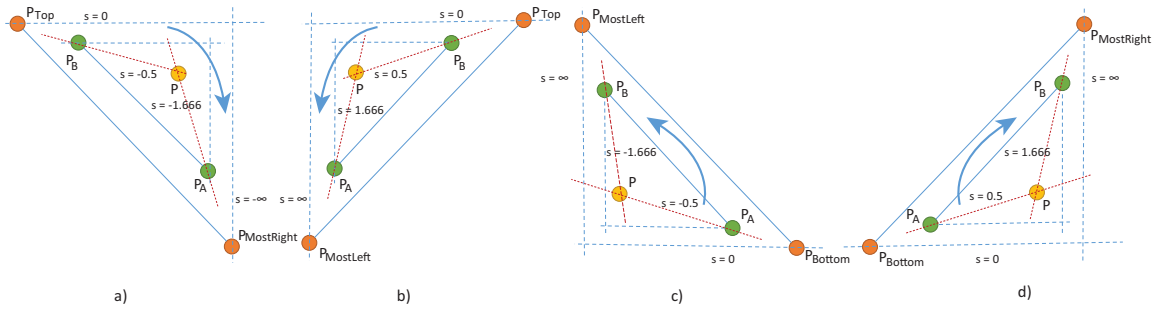


Figura 5.5: Rotación de la pendiente por cuadrante

5.2.2 Algoritmo Symmetric-Hull

La supresión de una serie de puntos, antes de comenzar con las operaciones más complejas, no es una idea nueva, pues ha sido utilizada desde la heurística de Akl-Toussaint [Akl and Toussaint, 1978] y ha sido mejorada constantemente. Siguiendo esta idea, existen aproximaciones que toman los cuatro puntos extremos del máximo paralelogramo inscrito, después los eliminan y finalmente realizan una serie de operaciones [Xiangyu et al., 2006] [Singh and Singh, 2013]. También existen aproximaciones incrementales que usan el máximo paralelogramo inscrito, como región inicial, y operan sobre los puntos restantes para llegar a la forma final de la cubierta convexa [Renwu et al., 2011] [Souviron, 2013]. Sin embargo, todas estas propuestas aún fallan, pues separan la eliminación de puntos de la creación de la cubierta convexa. Además, al operar de forma intensiva sobre el conjunto inicial de puntos, estos algoritmos pierden tiempo considerable y muchos de ellos usan la función de orientación como operación principal, ocasionando cuellos de botella al momento de eliminar puntos que no pertenecen a la cubierta convexa, dado que se generan más operaciones, tanto algebraicas como de pilas.

Una alternativa de la función de orientación es el uso de la pendiente, pero se suele descartar porque es necesario calcular dos pendientes y comparar sus resultados. Por lo tanto, se requieren cuatro sustracciones y dos divisiones, además de un control para evitar errores numéricos. En comparación con las cinco restas y dos multiplicaciones que requiere la función de orientación, los tiempos de ejecución terminarían siendo muy cercanos. Con el fin de evitar esta situación, en el presente trabajo se usan las observaciones expuestas en la Subsección 5.2.1. Para implementarlas, se requiere una pila de puntos candidatos a ser parte de la cubierta convexa por cada cuadrante y un cambio en la estructura de los puntos, pues cada punto $P(x, y)$ se convierte en $P(x, y, s)$, donde s es la pendiente entre el punto P y el último punto en la pila correspondiente.

Recordando las observaciones de la Subsección 5.2.1, es necesario ordenar D, antes de analizar los puntos, para obtener un arreglo D_S ordenado lexicográficamente por la coordenada y. Enseguida, se hará una búsqueda de los puntos extremos P_{Top}, P_{Bottom}, $P_{MostRight}$ y $P_{MostLeft}$ y se iniciarán cuatro pilas: T_1, T_2, T_3 y T_4. Cada pila corresponde

al cuadrante marcado por el número de su subíndice, cuyos valores iniciales serán los puntos extremos desde donde se inicia el recorrido, i.e., P_{Top} para los cuadrantes Q_1 y Q_2, dado que el recorrido será de arriba hacia abajo, y P_{Bottom} para los cuadrantes Q_3 y Q_4, pues el recorrido se hará de abajo hacia arriba, quedando como lo indican las Igualdades 5.5:

$$\begin{aligned} T_1 &= P_{Top} \\ T_2 &= P_{Top} \\ T_3 &= P_{Bottom} \\ T_4 &= P_{Bottom} \end{aligned} \tag{5.5}$$

Los puntos extremos siempre son parte de la cubierta convexa, por lo que este primer valor de las pilas nunca será removido. Cabe recordar que el recorrido se hace de arriba hacia abajo para Q_1 y Q_2, y de abajo hacia arriba para Q_3 y Q_4, ya que se busca analizar únicamente los puntos que están fuera del máximo paralelogramo inscrito R, cuyos vértices son dichos puntos extremos. Dada la lógica de SymmetricHull, si se recorriera D_S en un sólo sentido, se verificarían los puntos internos de R, en lugar de los externos, en los dos últimos cuadrantes analizados. Así, en un recorrido de abajo hacia arriba, los cuadrantes superiores serían mal analizados, mientras que en un recorrido de arriba hacia abajo, los cuadrantes perjudicados serían los inferiores.

Primeramente, se hace un recorrido sobre D_S, de manera que cada punto P en D_S es analizado, utilizando los puntos extremos para determinar a cuál de los cuatro cuadrantes pertenece. Dada la pertenencia de P a uno de los cuadrantes, es posible aplicar las reglas de la Observación 1 (ver Subsección 5.2.1) la cual establece que un punto candidato únicamente puede existir en la región definida por sus vecinos en una pila de candidatos. En este caso, dicha región puede ser especificada con base en el último punto, denominado P_{ant}, de la pila de candidatos respectiva y el punto extremo horizontal correspondiente al cuadrante (i.e., $P_{MostRight}$ para los cuadrantes Q_1 y Q_4, y $P_{MostLeft}$ para Q_2 y Q_3).

La cubierta convexa del cuadrante Q_1 se construye de arriba hacia abajo y de izquierda a derecha, comenzando por P_{Top} hasta alcanzar $P_{MostRight}$; de la misma manera, la cubierta convexa de Q_2 se edifica de arriba hacia abajo y de derecha a izquierda hasta alcanzar $P_{MostLeft}$; para el caso de Q_3, la cubierta convexa se erige de abajo hacia arriba y de derecha a izquierda, empezando por P_{Bottom} hasta alcanzar $P_{MostLeft}$; finalmente, la cubierta convexa de Q_4 se fabrica de abajo hacia arriba y de izquierda a derecha hasta alcanzar $P_{MostRight}$.

Dado este sentido de la construcción de cubiertas convexas por cuadrante, sólo es necesario saber si P (i.e., el punto en proceso de análisis) está a la izquierda o a la derecha de P_{ant}, dependiendo del cuadrante, para determinar si P está dentro de la región que describen las reglas de la Observación 1 (ver Subsección 5.2.1), pues P no podría rebasar los puntos extremos en ninguna iteración. Puesto que se está

haciendo el recorriendo de D_S, que es un arreglo ordenado lexicográficamente por la coordenada y, se sabe que el punto P tiene, por lo menos, la misma altura que P_{ant}, o bien se encuentra más avanzado en el eje de las ordenadas. De esta manera, es posible traducir las reglas de la Observación 1 (ver Subsección 5.2.1) como una simple comparación de la coordenada x de P, respecto a la coordenada x_{ant} de P_{ant}, según el cuadrante, como lo indican las Inecuaciones 5.6:

$$\begin{aligned} Q_1 &: x > x_{ant} \\ Q_2 &: x < x_{ant} \\ Q_3 &: x < x_{ant} \\ Q_4 &: x > x_{ant} \end{aligned} \tag{5.6}$$

Esta simple comparación descarta una gran cantidad de puntos, ya que elimina la necesidad de analizar los puntos interiores. El siguiente paso en el recorrido sobre D_S es calcular la pendiente entre P y P_{ant}, asignándola como la propiedad s de P. Enseguida, s es comparada con la pendiente s_{ant} del punto P_{ant}, según las reglas de la Observación 2 (ver Subsección 5.2.1). De esta manera, se establecen las Condiciones 5.7 para cada cuadrante:

$$\begin{aligned} Q_1 &: s > s_{ant} \\ Q_2 &: s < s_{ant} \\ Q_3 &: s < s_{ant} \\ Q_4 &: s > s_{ant} \end{aligned} \tag{5.7}$$

Las reglas de la Observación 2 (ver Subsección 5.2.1) permiten conocer el sentido en el que se mueve la pendiente, en un cuadrante, del exterior al interior. Este es el mismo movimiento que sigue la cubierta convexa, pasando de la posición más horizontal a la más vertical. Al utilizar las Condiciones 5.7, se pretende verificar si se obtuvo una pendiente más horizontal que la anterior; si este es el caso, significa que P_{ant} no pertenece a la cubierta convexa y que debe ser eliminado de la pila de candidatos T del cuadrante correspondiente. Este proceso se repite en un bucle, en el que se compara la pendiente s_{ant} del nuevo tope de la pila (el nuevo P_{ant}) con la pendiente s de P (entre P y P_{ant}); este bucle termina cuando s es más vertical que s_{ant} y, finalmente, se inserta P en la pila T correspondiente.

Si bien esta operación parece similar a la de los algoritmos clásicos, como Monotone Chain [Andrew, 1979], el cambio en la representación de $P(x, y)$ a $P(x, y, s)$ y la comparación de pendientes entre puntos consecutivos permiten disminuir drásticamente el número de operaciones algebraicas, pues cada operación de eliminación sólo necesita dos restas y una división. Además, gracias a las Condiciones 5.6, SymmetricHull realiza la comparación de pendientes, operando únicamente sobre el conjunto de candidatos a ser parte de la cubierta convexa, lo cual disminuye el número de operaciones

necesarias para el manejo de pilas. En contraste, los algoritmos clásicos utilizan la función de orientación, la cual requiere cinco restas y dos multiplicaciones y debe ser aplicada sobre todos los puntos del conjunto de entrada.

Como se mencionó en la asignación de valores iniciales a las pilas (ver Igualdades 5.5), cada una de ellas inicia con un valor de P_{Top} o P_{Bottom}, según el cuadrante. Sin embargo, al no haber un punto anterior en cada pila, no es posible asignar un valor a la propiedad s de este punto inicial. Por lo tanto, para evitar que estos puntos de partida sean descartados a causa de la pendiente, esta propiedad es inicializada mediante las Asignaciones 5.8:

$$\begin{aligned} T_1 &: s_0 = -\infty \\ T_2 &: s_0 = \infty \\ T_3 &: s_0 = \infty \\ T_4 &: s_0 = -\infty \end{aligned} \tag{5.8}$$

Cabe destacar que, gracias a las Condiciones 5.6, la comparación entre pendientes no conllevará a errores numéricos, aunque se trate de una división. Estas condiciones permiten garantizar que P (i.e., el punto en proceso de análisis) se encuentra más avanzado en el eje de las abscisas que el tope de la pila de candidatos, con respecto a la dirección de la cubierta convexa. De esta manera, el denominador de la pendiente nunca será cero. El único caso en el que dos puntos, que forman parte de la cubierta convexa, tendrán la misma coordenada x ocurre cuando el recorrido alcanza un punto, cuya coordenada x es igual a la del extremo horizontal del cuadrante. En este caso, si aún existen puntos por agregar a la cubierta convexa del cuadrante, dichos puntos estarán alineados sobre la línea vertical descrita por P y el extremo horizontal del cuadrante (i.e., $P_{MostRight}$ para los cuadrantes Q_1 y Q_4 o $P_{MostLeft}$ para los cuadrantes Q_2 y Q_3).

Dicho de otra manera, todos los puntos que pertenecen a la cubierta convexa y que aún no han sido agregados tendrán la misma coordenada x que P y el extremo horizontal. En este caso podrían ocurrir errores numéricos por división entre cero, pero como todos estos puntos caen en la misma línea vertical pueden ser omitidos, dado que no generan nueva información sobre la cubierta convexa. Por lo tanto, SymmetricHull detiene la búsqueda de puntos cuando se alcanza uno que tiene la coordenada x extrema del cuadrante. Una vez que el recorrido sobre D_S ha terminado, cada pila T tiene la parte de la cubierta convexa correspondiente a su cuadrante. Finalmente, basta con concatenar las pilas para obtener la cubierta convexa total. El proceso que sigue SymmetricHull se resume en el Algoritmo 5.

Input: D
D_S=OrdenarLexicograficamente(D);
[P_{Top}, P_{Bottom}, $P_{MostRight}$, $P_{MostLeft}$] =BuscarExtremos();
InicializarPilas() ; // usando 5.5 y 5.8
foreach $P \in D_S$ **do**
  if *P.x != extremoHorizontal()* **then**
    ClasificarPorCuadrante(P);
    if *Condiciones 5.6 == true* **then**
      $P.s = (P.y - P_{ant}.y)/(P.x - P_{ant}.x)$;
      while *Condiciones 5.7 == true* **do**
        Pop();
        $P.s = (P.y - P_{ant}.y)/(P.x - P_{ant}.x)$;
      end
      Push(P);
    end
  end
  else
    Break;
  end
end
Concatenar(T_1, T_2, T_3, T_4);

Algoritmo 5: SymmetricHull

5.3 Descomposición por aproximación convexa

El proceso de descomposición convexa puede ser un proceso difícil para una imagen. Es por ello que se necesitan las herramientas matemáticas adecuadas para que el proceso sea lo más eficiente posible. En nuestro caso, hemos desarrollado el Algoritmo 5 que podemos aplicar sobre el conjunto de grafos H que hemos obtenido, del proceso descrito en la Sección 5.1, para obtener los segmentos convexos de un *sketch* con sólo modificar ligeramente el Algoritmo 5.

5.3.1 Segmentación convexa sin huecos

El caso básico de la descomposición convexa es la descomposición de un grafo $G = (V, E)$ con $V = v_1, v_2, \ldots, v_m$ sin huecos. Para ello usamos el Algoritmo 5 sobre la lista de vértices V, bajo la premisa de obtener las cubiertas convexas de los diferentes segmentos convexos que forman al objeto representado por el grafo G; este algoritmo tiene un tiempo de ejecución $O(nlogn)$, sin embargo este tiempo de ejecución se debe al ordenamiento lexicográfico necesario. Gracias al procesamiento descrito en la Sección 5.1, la lista de vértices V ya se encuentra ordenada lexicográficamente por la coordenada y, por lo que el primer paso del Algoritmo 5 no es necesario (la aplicación

del Algoritmo 5 tiene un tiempo de $O(n)$). El siguiente paso sería conocer los puntos extremos, pero trabajamos bajo la premisa de que un objeto está formado por varios segmentos convexos, cada uno con sus puntos extremos. Esta dificultad es fácilmente sorteable si se consideran los puntos de inflexión, en lugar de los puntos extremos.

Es importante aclarar que nuestro objetivo no es obtener la cubierta convexa, sino más bien verificarla, ya que las aristas del grafo G representan los bordes de la imagen, i.e., ya no hay puntos interiores, pero siguiendo el Algoritmo 5 es posible encontrar los puntos de inflexión y hacer la separación convexa.

El proceso inicia con el vértice más alto en el grafo, i.e., aquel con la coordenada y mayor. Este vértice es forzosamente el punto P_{Top} de, al menos, un segmento convexo. También, sabemos que se tienen dos direcciones posibles, izquierda y derecha, y que por lo menos P_{Top} está conectado a dos vértices. Entonces, se toman dos vértices consecutivos en la horizontal (ya que P_{Top} puede estar conectado a más vértices) y se comienza el recorrido desde P_{Top}. El vértice que esté más a la derecha pertenece al cuadrante Q_1 y se le aplican las reglas 5.6 y 5.7 correspondientes a dicho cuadrante; el vértice localizado más a la izquierda pertenece al cuadrante Q_2 y se aplican las reglas 5.7 para ese cuadrante.

Con el mismo conjunto de reglas 5.7 es posible conocer los puntos de inflexión, e.g., en el cuadrante Q_1 pueden ocurrir tres casos por cada vértice analizado en el grafo G: 1) el nuevo vértice pertenece a la cubierta del cuadrante Q_1; 2) el nuevo vértice está a la derecha del último vértice analizado, según el Algoritmo 5; esto lo convierte en candidato a pertenecer a la cubierta, pero su pendiente es mayor que la anterior medida, lo cual eliminaría el punto previamente analizado; sin embargo, como todos los vértices del grafo G pertenecen a una cubierta, esto significa que se ha encontrado un vértice que pertenece a otra zona convexa, por lo tanto es un punto de inflexión; y 3) el nuevo vértice analizado está abajo y a la izquierda; en el Algoritmo 5 significaría que se debe descartar este punto, pero como se sabe que todo vértice pertenece a una cubierta convexa, este punto sólo es la continuación de la cubierta en el cuadrante Q_4; en este caso, se ha encontrado un punto de inflexión sobre la misma cubierta convexa y es necesario continuar el análisis, aplicando las reglas para el cuadrante Q_4.

En cuanto al análisis del cuadrante Q_4, también se tienen tres casos: 1) el vértice analizado está a la izquierda y abajo, y tiene una pendiente menor que el último vértice, por lo que es parte de la cubierta del cuadrante Q_4; 2) el vértice está abajo y a la izquierda, pero tiene una pendiente mayor, indicando un giro en la dirección contraria, por lo tanto se trata de un punto de inflexión; y 3) el vértice está a la derecha y abajo, por lo que es un punto de inflexión, ya que pertenece a una cubierta convexa, pero no es parte de Q_4.

El mismo análisis que se hizo para Q_1 se hace para el cuadrante Q_2, comenzando desde el vértice que se considera P_{Top}, sólo que en sentido inverso en el eje x. Los tres casos para cada vértice analizado en Q_2 son los siguientes: 1) si está a la izquierda y abajo, y su pendiente es mayor que la pendiente del vértice anterior, entonces es un vértice que forma parte de la cubierta de Q_2; 2) si está a la izquierda y abajo, pero

su pendiente es menor, entonces se trata de un punto de inflexión que no pertenece a la cubierta de Q_2; 3) el vértice analizado está a la derecha y abajo del último vértice analizado, por lo es un punto de inflexión que pasa a ser parte del siguiente segmento de puntos de la cubierta convexa, i.e, pertenece a Q_3.

Con el cuadrante Q_3 suceden los mismos casos que para el cuadrante Q_4 pero en sentido inverso en el eje x: 1) el vértice actualmente analizado está a la derecha y abajo, y tiene una pendiente menor que el último vértice, por lo que es parte de la cubierta de Q_3; 2) el vértice está abajo y a la derecha, pero tiene una pendiente menor, indicando un giro en la dirección contraria, por lo tanto es un punto de inflexión que no pertenece a Q_3; y 3) el vértice está a la izquierda y abajo, por lo que es un punto de inflexión, ya que pertenece a una cubierta convexa, pero no es parte de Q_3.

Ahora que ya conocemos cómo encontrar los puntos de inflexión de ambos lados de la cubierta desde P_{Top} (por la derecha recorriendo Q_1 y Q_4, y por la izquierda recorriendo Q_2 y Q_3) el algoritmo para encontrar los puentes (*bridges*) que cortan una figura en segmentos convexos es sumamente sencillo. El proceso inicia creando un nuevo grafo $G_a = (V_a, E_a)$ que será convexo y estará separado de todas las figuras, por lo que cada vértice $v_i \in V_a$ sólo tiene conexiones a dos vértices y además $|E_a| = |V_a|$ si no se cuentan las repeticiones. El recorrido comienza por ambos lados desde P_{Top}. Para cada lado, se agrega cada vértice v_i visitado al nuevo grafo G_a, siendo P_{Top} el vértice inicial. Para cada vértice visitado se agrega una arista, indicando su relación con el último vértice analizado en este lado del recorrido, así todos los vértices quedarán enlazados a un vértice anterior y uno posterior. El recorrido de cada lado se detendrá al encontrar un punto de inflexión que no sea continuación hacia Q_4, si se viene desde Q_1, o una continuación a Q_3, si se está analizando desde Q_2, i.e., los vértices que han sido descritos como puntos de inflexión y que pertenecen a otras cubiertas. Llamamos a estos puntos de inflexión P_L para el punto de inflexión del recorrido por la izquierda (Q_3 y Q_3) y P_R para el recorrido por la derecha (Q_1 y Q_4).

Conocidos los dos puntos de inflexión de un segmento convexo, lo lógico sería cerrar el grafo G_a creando una arista entre los dos puntos de inflexión, sin embargo, existe un riesgo. Aunque ambos lados de la cubierta sean convexos, es posible que al conectarlos no lo sean, como muestra la Figura 5.6, pues la diferencia en el avance en el eje y puede provocar concavidades. Estas sólo se presentan si se unen puntos de inflexión que se identificaron en las cubiertas de Q_3 y Q_4, ya que si el punto de inflexión viene de la cubierta de Q_1, entonces la nueva unión representa la totalidad de la cubierta de Q_4 y el grafo G_a es convexo. Lo mismo sucede si el punto de inflexión está en Q_2, ya que la arista generada representaría en su totalidad a la cubierta de Q_3. Estos mismos escenarios pueden observarse en la Figura 5.6. En ambos casos es posible crear la arista entre los puntos de inflexión, haciendo un corte (*bridge*) y creando un segmento convexo. Si se tiene el caso de que los puntos de inflexión se encuentren en Q_3 y Q_4, entonces es necesario verificar las pendientes para saber si los puntos de inflexión no forman una concavidad: se toma el punto de inflexión en Q_3 y en Q_4 y se obtiene su pendiente s_{34}; esta es comparada con la pendiente que el punto de

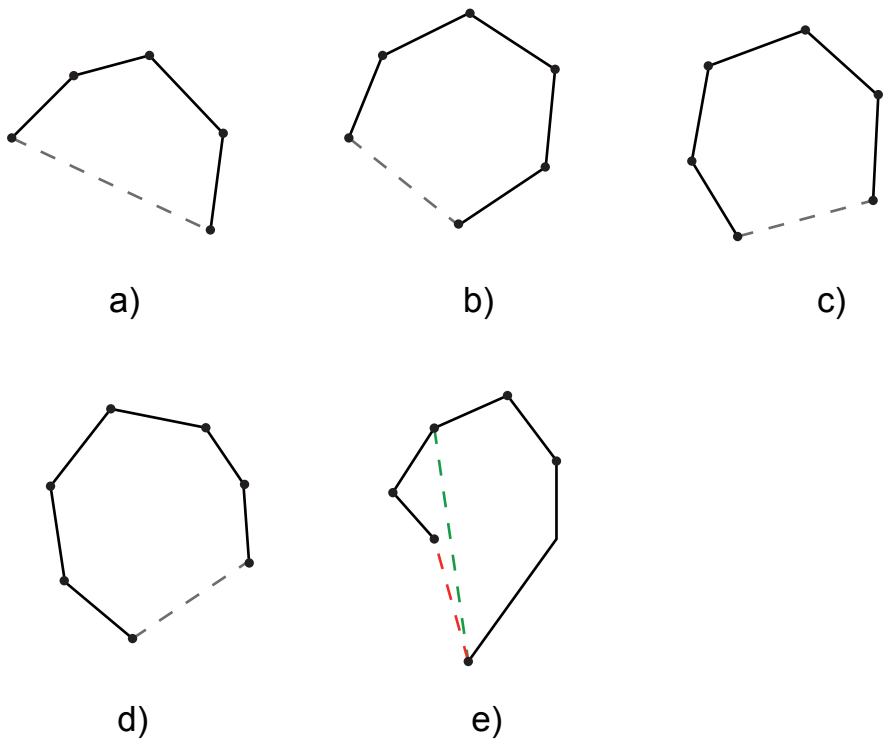


Figura 5.6: En a) se puede ver el caso donde los puntos de inflexión están en Q_1 y Q_2. En b) los puntos de inflexión están en Q_2 y Q_4, mientras en c) están en Q_1 y Q_3. Para d) los puntos de inflexión están en Q_3 y Q_4 sin que existan obstrucciones. En todos los casos anteriores la conexión es directa, pero en el caso de e) se tienen que comparar pendientes para determinar el mejor vértice para realizar el corte.

inflexión en Q_3 tiene respecto a su vértice predecesor s_3; se debe cumplir que $s_{34} \geq s_3$, de no ser así, en lugar de tomar el punto de inflexión en Q_4, se toma su predecesor y se verifica nuevamente la condición, retrocediendo sobre los vértices analizados en Q_4 hasta que se cumpla la condición y creando una arista entre el punto de inflexión en Q_3 y el vértice en Q_4 que satisface la condición. Lo anterior se cumple si el punto de inflexión en Q_3 tiene una coordenada y menor a la del punto de inflexión en Q_4; en caso contrario, el recorrido sobre los predecesores se hace sobre Q_3, en lugar de Q_4.

Los casos analizados en el párrafo anterior son aquellos en los que se crea un corte (*bridge*), pero también es posible recorrer todo el segmento y que este sea convexo. Por esta razón, es necesario marcar los vértices visitados para no volver a procesarlos.

Cuando se ha creado un corte, entonces se debe iniciar el proceso nuevamente, tomando $P_{Top} = MAX_y(\omega)$ donde ω es el conjunto de los vértices no visitados en G y los vértices que forman el extremo del corte (*bridge*) tales que aún tengan aristas que apunten a vértices no visitados. El proceso termina cuando el conjunto $|\omega| = 0$, i.e., todos los vértices han sido visitados. Para una mayor claridad en cómo este proceso

funciona, refiérase al Algoritmo 6.

En el Algoritmo 6, se manejan los pseudónimos *next* y *prev* para definir las aristas que mejor cumplen la convexidad, pero se debe tener en consideración que esta representación favorece la creación de G_a, donde cada vértice solo puede tener dos aristas, una apuntando al vértice anterior y otra al posterior. Sin embargo, en V, el mismo vértice puede tener varias aristas, por lo que *prev* y *next* deben ser escogidos como aquellos con la mejor convexidad, a partir de la lista de aristas del vértice. Para clarificar cómo funciona el Algoritmo 6, la Figura 5.7 muestra, paso a paso, cómo un grafo es segmentado en sus componentes convexos.

5.3.2 Segmentación convexa con huecos

Para el desarrollo de los procesos de clasificación y ajuste de primitivas tridimensionales de este trabajo, no es necesario tomar en cuenta huecos en las figuras, sin embargo, el desarrollo del Algoritmo 6 nos ha permitido crear una versión del mismo que es capaz de crear segmentos convexos en figuras que presentan huecos. Esta versión tiene un tiempo de ejecución competitivo y enriquece el estado del arte, pues permite obtener un número de piezas cercano al mínimo.

El algoritmo para la segmentación convexa en figuras que permiten huecos, utiliza como base el Algoritmo 6 de forma iterativa, analizando el espacio entre un hueco y una figura convexa (que representa el resto de los huecos en una sola agrupación convexa) lo que permite obtener zonas convexas más grandes que si se tratara de analizar todos los huecos a la vez.

Sea $G_a = (V_a, E_a)$ un grafo que representa un segmento convexo y sea $A = A_1, A_2, \ldots, A_n$ una serie de grafos que representan los huecos dentro de G_a, donde $A_i = (V_i, E_i)$. Dado que G_a es convexo y no está conectado a otros segmentos, entonces V_a por sí mismo representa la cubierta convexa, mientras que los huecos A_i pueden o no ser convexos. El primer paso es obtener los extremos de G_a (i.e., P_{Top}, P_{Bottom}, $P_{MostRight}$ y $P_{MostLeft}$). Por el procesamiento hasta ahora llevado, G_a ya está lexicográficamente ordenado por la coordenada y, además de que sus puntos de inflexión son conocidos, por lo que es fácil obtener sus puntos extremos. El segundo paso es obtener, mediante el Algoritmo 5, la cubierta convexa de A que se denota como CH_A. Después, se conectan los puntos extremos de G_a con los puntos extremos de CH_A, lo que dará como resultado una serie de regiones $R_E = (R_{E1}, R_{E2}, R_{E3}, R_{E4})$ (de hecho pueden ser sólo tres regiones cuando uno de los puntos extremos se repite, pero el procesamiento es exactamente el mismo). A dichas regiones se les aplicará el Algoritmo 6 para obtener los segmentos convexos, como se muestra en los pasos de los incisos de a) a d) en la Figura 5.8.

El resto del problema consiste en encontrar los segmentos convexos que se forman entre los huecos, por lo que se hace un procesamiento iterativo, retirando de A el grafo A_i que se encuentra más a la izquierda (es posible usar cualquier dirección,

```
Input: Graph G=(V,E)
ω = V;
while |ω|! = 0 do
    P_Top = Max_y(ω);
    P_L = Min_x(P_Top.next, P_Top.prev);
    P_R = Max_x(P_Top.next, P_Top.prev);
    while !isInflexionPoint(P_L) do
        G_a.add(P_L);
        P_L.visited = true;
        P_L = P_L.next;
    end
    while !isInflexionPoint(P_R) do
        G_a.add(P_R);
        P_R.visited = true;
        P_R = P_R.next;
    end
    if isIntoQ_1(P_R) then
        P_L.next = P_R ;
        P_R.next = P_L ;
    else if isIntoQ_2(P_L) then
        P_L.next = P_R;
        P_R.next = P_L;
    else
        s = (P_L.y − P_R.y)/(P_L.x − P_R.x);
        if P_L.y > P_R.y then
            while s ≤ P_L.s do
                G_a.remove(P_R);
                P_R.visited = false;
                P_R = P_R.prev;
            end
        else
            while s ≤ P_L.s do
                G_a.remove(P_L);
                P_L.visited = false;
                P_L = P_L.prev;
            end
        P_L.next = P_R ;
        P_R.next = P_L ;
    end
    ω = AllNoVisited(ω);
    if |P_L.edges| > 2 then
        ω+ = P_L;
    if |P_R.edges| > 2 then
        ω+ = P_R;
    H.add(G_a);
end
return H;
```

Algoritmo 6: Segmentación convexa

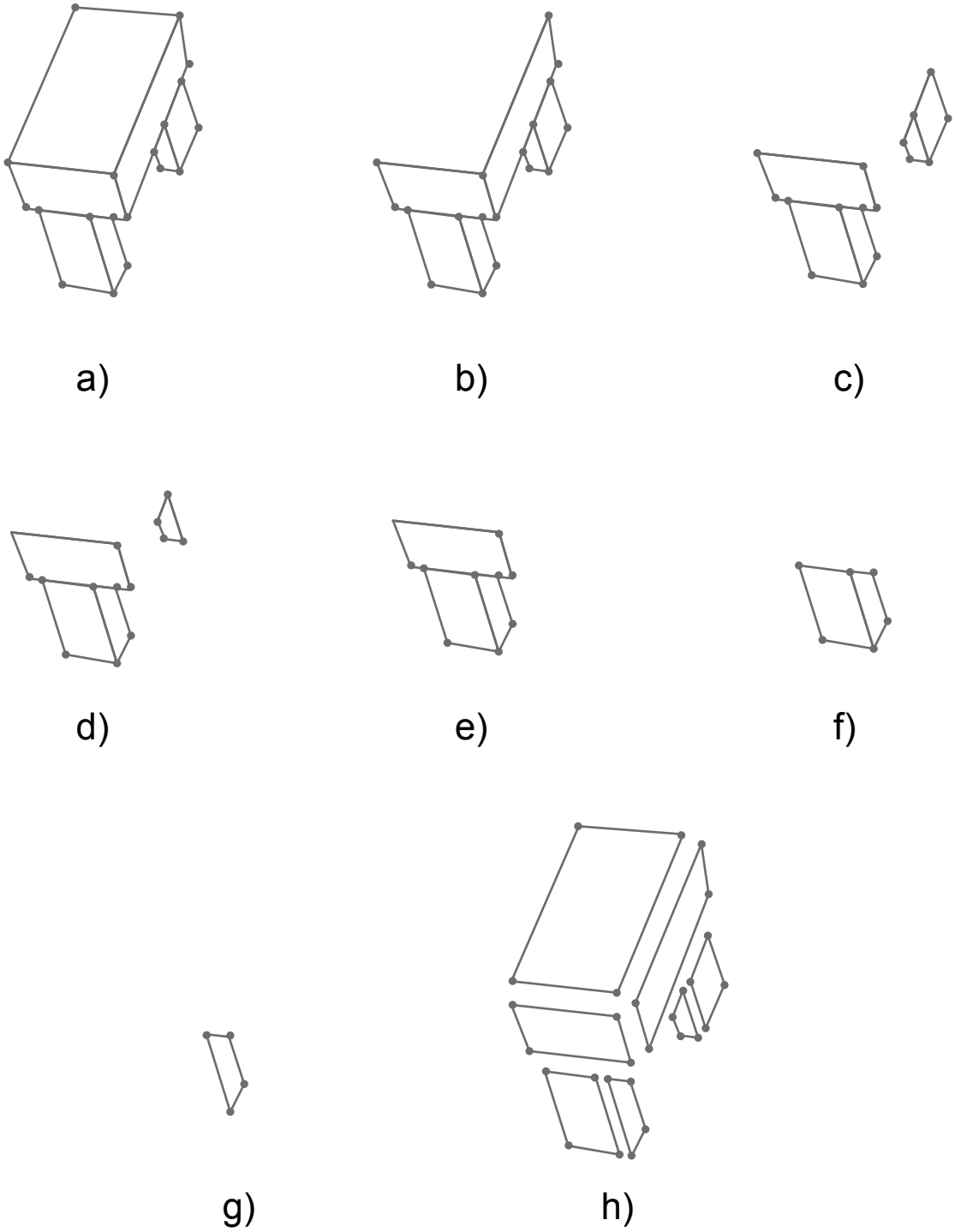


Figura 5.7: En a) se puede observar el grafo original, posicionando los vértices en las mismas posiciones que los puntos que representa, de esta manera de b) a g) se muestra cómo los segmentos son separados. El conjunto H es mostrado en h).

pero usamos la izquierda por conveniencia). Con el resto de A se forma de nuevo una cubierta convexa, la cual será restada de la cubierta convexa anterior, dando como resultado una región o serie de regiones que se irán agregando a un conjunto de grafos $T = (T_1, T_2, \ldots, T_m)$. A continuación, se remueve de A el grafo A_i con la posición más a la izquierda y se obtiene la nueva cubierta convexa, creando nuevas regiones que se agregaran a T, como se muestra en la Figura 5.8 en los incisos e) y f).

El último paso del Algoritmo 7 consiste en aplicar el Algoritmo 6 sobre cada elemento de T. De esta manera, se obtienen los segmentos convexos en las regiones entre los huecos, como se muestran en los incisos g) a k) de la Figura 5.8. El proceso completo de la segmentación convexa en figuras con huecos se puede observar en el Algoritmo 7.

Input: Graph $G_a = (V_a, E_a)$, Holes $A = (A_1, A_2, ..., A_n)$
$[P_{Top}, P_{Bottom}, P_{Mostright}, P_{MostLeft}] = SearchExtremes(G_a)$;
$CH_A = ConvexHull(A)$;
$[PCH_{Top}, PCH_{Bottom}, PCH_{Mostright}, PCH_{MostLeft}] = SearchExtremes(CH_A)$;
$T_1 = AllVertexAmong(P_{Top}, PCH_{Top}, P_{MostRight}, PCH_{MostRight})$;
$T_2 = AllVertexAmong(P_{Top}, PCH_{Top}, P_{MostLeft}, PCH_{MostLeft})$;
$T_3 = AllVertexAmong(P_{Bottom}, PCH_{Bottom}, P_{MostRight}, PCH_{MostRight})$;
$T_4 = AllVertexAmong(P_{Bottom}, PCH_{Bottom}, P_{MostLeft}, PCH_{MostLeft})$;
for *i=1 to n* **do**
  $L = Min_X(A)$;
  $A = A - L$;
  $CH_{aux} = ConvexHull(A)$;
  $T.add(AllRegionsFormedIn(CH_A - CH_{aux} - L))$;
end
for *i=1 to* $|T|$ **do**
  $H.add(SegmentacionConvexa(T_i))$;
end
return H;

Algoritmo 7: Segmentación convexa en figuras con huecos

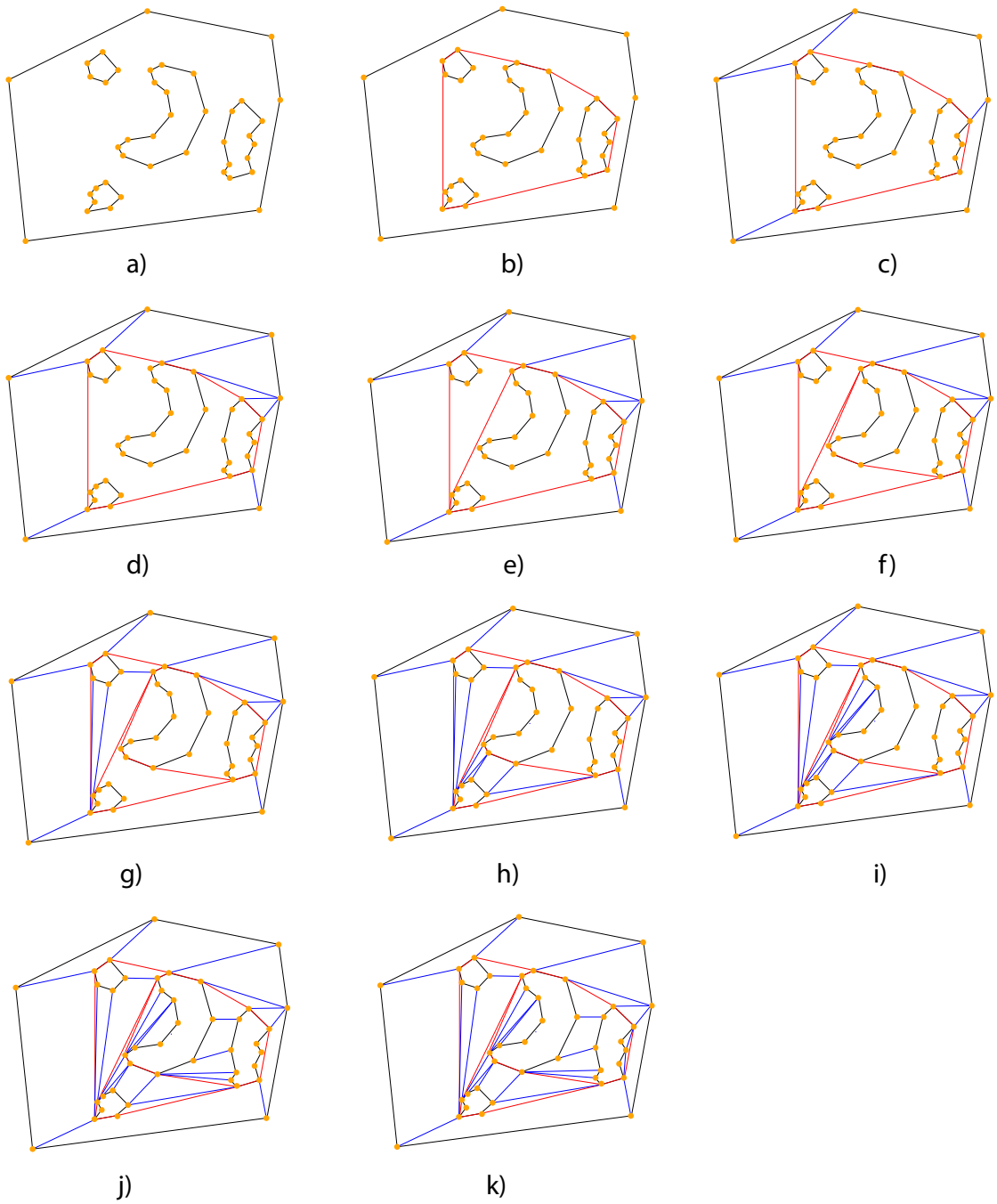


Figura 5.8: En a) se observa el grafo original con huecos también representados por grafos. En b) se puede ver cómo se crea la cubierta convexa del conjunto de huecos A. En c) y d) se observa la segmentación de la zona entre la cubierta exterior y la cubierta convexa de los huecos CH_A, primero en cuatro partes y después aplicando el Algoritmo 6. De e) a g) se aprecia la separación en regiones al aplicar el Algoritmo 5 iterativamente, quitando el hueco más izquierdo cada vez. Desde h) hasta k) se observa la separación convexa de las zonas obtenidas al aplicar iterativamente el Algoritmo 5.

Capítulo 6

Ajuste de primitivas tridimensionales

Todo el proceso hasta ahora presentado tiene la finalidad de facilitar la labor de este capítulo: el ajuste de primitivas tridimensionales; esto es, tomar las figuras tridimensionales básicas y ajustarlas a las formas convexas obtenidas en el Capítulo 5, definiendo qué figura empata mejor a la cubierta convexa y cuáles valores de esta figura son los que mejor ajustan a los puntos de dicha cubierta. Siguiendo con la metodología propuesta, utilizaremos técnicas de ajuste de primitivas tridimensionales y calibración de los valores mediante restricciones geosemánticas, que han sido utilizadas para la reconstrucción de *sketches* [Chen et al., 2013], [Shtof et al., 2013].

El trabajo presentado tiene por objeto la identificación de objetos hechos por el ser humano, usando como herramienta el ajuste de primitivas tridimensionales (*fitting*). Para esto, la primera tarea es definir las primitivas tridimensionales que serán utilizadas de entre la gran variedad posible. No obstante, la limitante de los "objetos hechos por el ser humano" acota el campo a sólo unas pocas primitivas tridimensionales, tal como lo sugieren [Figueiredo et al., 2017]. Según estos autores, los objetos funcionales de la vida diaria (excluyendo los decorativos) están compuestos sólo por las primitivas tridimensionales más sencillas, como son: cubos, cajas o prismas rectangulares, cilindros, esferas y conos, aunque también pueden aparecer toroides, prismas trapezoidales y pirámides, entre otros, pero en una medida mucho menor. Es por ello que este trabajo tomará únicamente las primeras cinco primitivas de la lista.

Dado que conocemos las primitivas tridimensionales que utilizaremos, queda por realizar las tareas de selección y ajuste. Esta última es ampliamente estudiada por la rama de Visión por Computadora, aplicándose gran variedad de técnicas como redes neuronales [Zou et al., 2017], campos aleatorios de Markov [Shao et al., 2012], gramáticas [Lau et al., 2011], optimización numérica y métodos evolutivos [Meng et al., 2013] y transformada de Hough [Borrmann et al., 2011]. Sin embargo, de entre todas ellas destacan los métodos basados en RANSAC [Bolles and Fischler, 1981], por su rapidez y eficiencia al ajustar primitivas tridimensionales, como ha sido demostrado en una gran

variedad de trabajos [Chen et al., 2015], [Oehler et al., 2011], [Ruiz et al., 2014]. La importante cantidad de propuestas, en el estado del arte, nos ayuda a decantarnos por una aproximación basada en RANSAC, además de que podemos obtener porcentajes de error en cada iteración, con base en la selección de puntos, a diferencia de otras propuestas, por ello durante la sección 6.1 mostraremos el uso de RANSAC como herramienta para el ajuste de primitivas tridimensionales que se completa con el uso del Langragiano aumentado, del que exploraremos sus posibilidades en la sección 6.2.

Una vez comprendido el uso de RANSAC y el Langragiano aumentado, es posible desarrollar las primitivas tridimensionales como una función objetivo para los procesos antes mencionados, lo cual se verá en la sección 6.3. Al tener sólo puntos de alta entropía (obtenidos del proceso de segmentación convexa en el capítulo 5), es posible usar esta propiedad para realizar la tarea de selección de primitivas, como se verá en la Sección 6.4.

Finalmente, la sección 6.5 se refiere a cómo utilizar las restricciones geosemánticas propuestas por [Shtof et al., 2013], tales como ortogonalidad, paralelismo, coplanaridad y colinearidad, entre otras, para lograr un ajuste fino de primitivas y completar los segmentos que no son visibles por la cámara.

6.1 Algoritmo RANSAC para el ajuste de primitivas tridimensionales

El problema al que nos enfrentamos en este momento es cómo ajustar los puntos de las cubiertas, que obtuvimos como resultado del proceso descrito en el Capítulo 5, a las primitivas tridimensionales conocidas. Una solución ingenua podría ser ocupar los puntos de frontera que ya tenemos para generar las ecuaciones. Por ejemplo, a partir de tres puntos sería fácil generar la ecuación de la esfera, resolviendo un sistema de ecuaciones simple. Para que esto funcione, los puntos deberían caer exactamente en la superficie de la esfera, lo cual pocas veces es cierto, pues nuestra mediciones no son exactas, ya sea por la discretización del espacio, por los errores y ruido en el sensor, o por el redondeo de valores en los procesos anteriores. Todos estos efectos provocarían que el sistema de ecuaciones no tuviera solución.

La mejor solución es utilizar un algoritmo de regresión para ajustar la mayoría de los puntos al mejor modelo posible. Frecuentemente, en la literatura científica, encontramos el uso de RANSAC como algoritmo de regresión para el ajuste de puntos a estructuras tridimensionales [D'Hondt et al., 2013], no sólo primitivas. Esto se debe a que es altamente resistente a valores que no caen dentro del modelo (*outliers*), i.e., los valores que caen dentro del modelo (*inliers*) dominan la solución, buscando un error mínimo, mientras otros algoritmos de regresión, como mínimos cuadrados, sufren de una fuerte desviación de resultados debida a los *outliers*. Como ejemplo de esto, nos referimos a la Figura 6.1, en la que en a) se pueden ver los puntos que se desean ajustar; para nosotros, como seres humanos, es fácil interpretar que la mayoría de

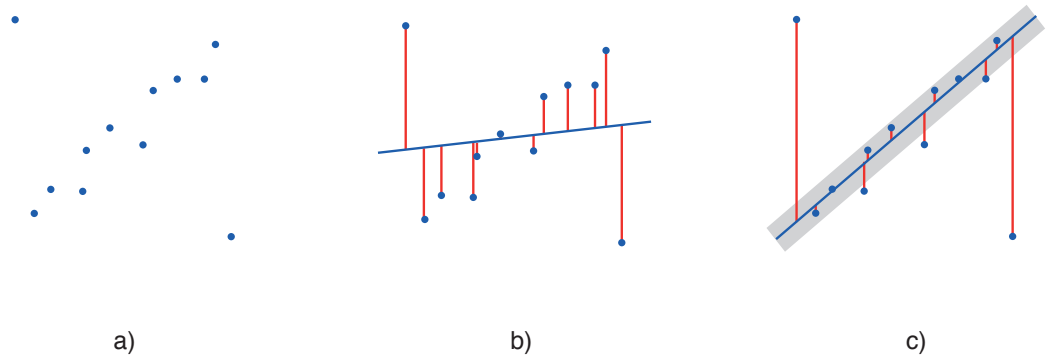


Figura 6.1: En a) observamos el conjunto de entrada s. En b) se muestra el resultado de la estimación de una línea a partir de s, usando mínimos cuadrados, mientras en c) se muestra el resultado de la misma estimación usando RANSAC.

los puntos se ajustan a una línea a 45 grados y que sólo hay dos puntos que parecen salir de esta descripción; si se pudiera hacer una interpretación en hardware, diríamos que son valores atípicos derivados de una mala medición del sensor, siendo esta una situación común en cualquier proceso de este tipo; sin embargo, para una computadora no es tan sencillo ese análisis deductivo, pues sólo se acatará a su programación, por lo que es necesario saber qué clase de resultado se requiere; si se programara una regresión por mínimos cuadrados, se obtendría una solución como la que aparece en b), la cual minimiza la distancia hacia el modelo por parte de todos los elementos del conjunto de entrada, pero ciertamente no es lo que se quiere; el resultado en c) es más parecido a lo que se espera y, de hecho, es el resultado de una regresión utilizando RANSAC, pues maximiza el modelo que tenga los mejores resultados respecto a la mayoría.

La descripción original de RANSAC [Bolles and Fischler, 1981] presenta un proceso genérico, que es capaz de adaptar un conjunto de entrada $s = [s_1, s_2, \ldots, s_v]$ a un modelo m. Según el algoritmo 8, el proceso para crear el modelo cambia para cada aplicación de RANSAC, debiéndose ajustar un parámetro n, que es el número de entradas que se necesita para crear un primer modelo. Por ejemplo, en la Figura 6.1 se intenta estimar una línea a partir del conjunto de entrada s, por lo que se puede generar el modelo a partir de dos puntos ($n = 2$). En el caso de los modelos para primitivas tridimensionales, estos serán explicados en la Subsección 6.3. Una vez que se tiene un modelo m se debe buscar el ajuste de la mayor cantidad de puntos de s (descartando los que se usaron para generar el modelo) a m; tratándose de mediciones con sensores, es casi seguro que no se obtendrán valores descritos exactamente por m, por lo que es necesario un valor de umbral t que representa el error tolerado entre la i-ésima entrada s_i y el valor arrojado por m para esa entrada. Se dice que una entrada $s_i \in I$, si y solo si $\|s_i - m(s_i)\| \leq t$; asimismo, se describe b como el número de entradas que deben existir en I para que un modelo sea considerado aceptable,

i.e., $|I| \geq b$. Lo anteriormente descrito se debe ejecutar iterativamente en k ocasiones, de forma que en cada nueva iteración se construye un nuevo modelo m y se evalúa si cumple la condición de aceptabilidad; en caso de cumplirla, se compara este modelo contra el mejor modelo hasta el momento (M), usando una variable auxiliar B que guarda el número de entradas que se ajustaron al modelo M; si $|I| \geq B$ entonces m es mejor que el mejor modelo encontrado hasta el momento (M) y se hace $M = m$ y $B = |I|$. El resultado final son los parámetros de M, que describen el mejor modelo que obtuvo el proceso.

Input: s, n, k, t, b
for i=*1 to* k **do**
  InliersIniciales=SeleccionarAleatoriamente(s, n);
  m=CrearModelo(InliersIniciales);
  $posiblesInliers$=NULL;
  foreach $s_i \in s \mid s_i \notin InliersIniciales$ **do**
    if $\|s_i - m(s_i)\| \leq t$ **then**
      $posiblesInliers$.Add(s_i);
    end
  end
  if $|posiblesInliers| \geq b$ **then**
    err=medidaError(m, $posiblesInliers$);
    if $err < Error$ **then**
      $M = m$;
      $Error = err$;
    end
  end
end
return M;

Algoritmo 8: RANSAC para propósitos generales

Si bien el algoritmo 8 podría parecer complicado a primera vista, lo cierto es que el poder de RANSAC radica en su simpleza. Daremos una breve explicación, partiendo del ejemplo de la Figura 6.1 y explicando los pasos para llegar de una nube de puntos a un modelo que describe la línea que pasa por la mayoría de los puntos. Para esto, es necesario hacer uso de la Figura 6.2, en cuyo inciso a) podemos observar la entrada s. Tratándose de una línea, se sabe que es posible construir un modelo $y = m_l x + b_l$, donde $m = \frac{y_2 - y_1}{x_2 - x_1}$, i.e., sólo depende de dos puntos para su construcción, lo que significa que $n = 2$. Los parámetros k, b y t serán estudiados a profundidad más adelante en esta sección, pero por el momento, para este ejemplo basta con situar $t = 0.5$, $k = 5$ y $b = 4$, así como establecer el método de medición del error como el conteo de *inliers*. En b) se muestra la primera iteración del algoritmo, donde se ha seleccionado aleatoriamente dos puntos del conjunto de entrada, los puntos llamados *InliersIniciales* y marcados en amarillo, produciendo una línea $y = -1$ que expande sus fronteras, gracias al umbral t, y que se nota con un área azul tenue alrededor

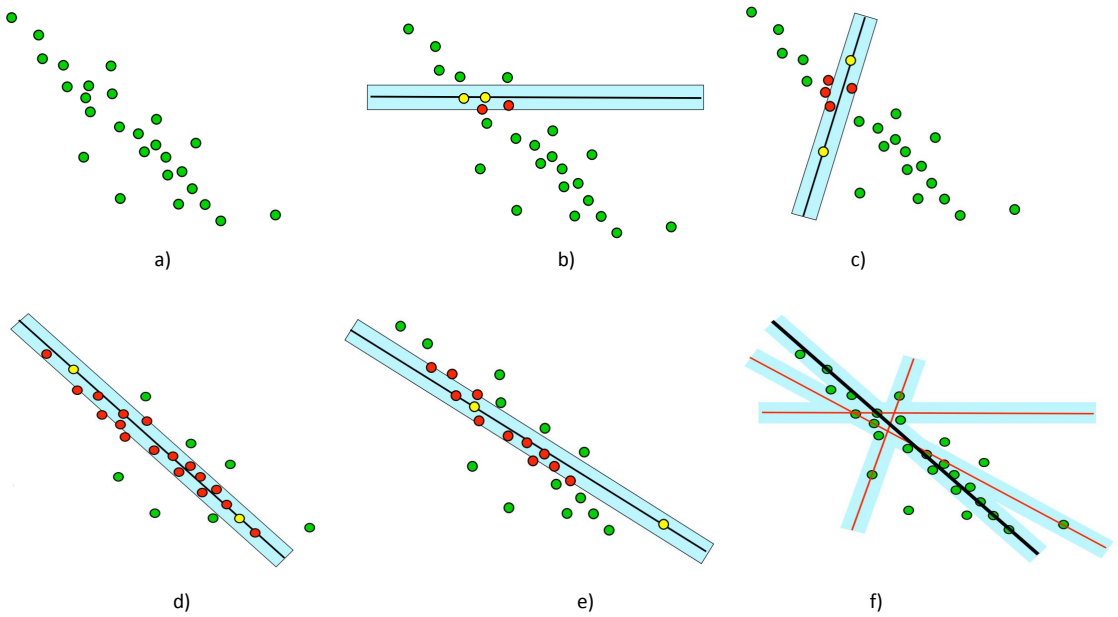


Figura 6.2: Modelo de la esfera para representarse como una función objetivo

de la línea; como se puede observar, el área azul sólo toca dos puntos, por lo que $|posiblesInliers| < b$, así que no es posible pasar a la siguiente etapa del algoritmo. En c) se escogen dos puntos distintos, dando como resultado una línea de la forma $y = 4x + 5$ que toca cuatro puntos más, gracias al umbral t, por lo que pasa a ser una línea candidata y se conservan estos valores, como B y M para la siguiente iteración d); los puntos iniciales dan como resultado una línea de la forma $y = -1x - 0.5$, que alcanza 17 puntos, por lo que pasa a ser el nuevo modelo M. Finalmente en e) se propone una línea $y = -0.8x - 1$ que toca 11 puntos, por lo que pese a que cumple la condición de modelo mínimo propuesto por b, no será el nuevo modelo M, pues el valor de B es 17. El resultado es que M es el mismo modelo del inciso d); como podemos ver en el inciso f), es el modelo que mejor se acomoda a la nube de puntos s propuesta.

RANSAC resulta tan resistente a los *outliers* como sea necesario (puede haber hasta 50 % de *outliers*), al ajustar adecuadamente los parámetros de b y k. Parámetros de b y k muy grandes darán como resultado un valor de M exacto, pero sumamente tardado; valores más pequeños podrían arrojar un resultado rápido, pero con poca exactitud. Si bien hay una gran cantidad de enfoques para ajustar adecuadamente estos parámetros, la mayoría toma en cuenta el conocimiento previo de la cardinalidad del conjunto (en nuestro caso, el conjunto de entrada es variable) y las condiciones del problema, por lo que en nuestro caso usaremos un enfoque más general para decidir el valor de k [Lacey et al., 2000].

6.1.1 Parámetro de iteración k

Sea e la probabilidad de escoger un *outlier*, entonces $1-e$ es la probabilidad de escoger un *inlier*, $(1-e)^q$ es la probabilidad de escoger q *inliers* iniciales y $1-(1-e)^q$ es la probabilidad de que, al menos, una entrada inicial sea *outlier*; si se eleva ese término al número de iteraciones ($(1-(1-e)^q)^k$), dará la probabilidad de que en k iteraciones no haya *outliers*, entonces $1-(1-(1-e)^q)^k)$ es la probabilidad de que, al menos, una muestra no sea *outlier*, la cual puede ser igualada con el porcentaje de asertividad que se desea obtener: $Pr = 1-(1-(1-e)^q)^k)$. Es posible resolver k de la última ecuación, así que el despeje se muestra en la Ecuación 6.1:

$$k = \frac{log(1-Pr)}{log(1-(1-e)^q)} \tag{6.1}$$

Ahora que se tiene la Ecuación 6.1, sólo se debe conocer Pr, e y q. El primero es un valor propuesto por nosotros; la mayoría de los algoritmos de ajuste de primitivas manejan $Pr = 0.95$ [Pluskal et al., 2010], por lo que nos hemos guiado por esta norma. El valor de e debe ser obtenido experimentalmente a partir de los datos del sensor y de factores ambientales, lo cual requiere de un arduo trabajo, pero afortunadamente en [Choo et al., 2014] el autor mide este error en varias iluminaciones y ambientaciones, dando un resultado de $e = 0.05$. El único valor que falta por determinar es q, este valor depende del conjunto de entrada, como se establece en [Bolles and Fischler, 1981], RANSAC genera un buen modelo incluso hasta con la mitad de *outliers* en s, por lo que podemos establecer q tal que $n \geq \frac{|s|}{2}$. El problema recae en la entrada, que es dinámica, pues el valor de $|s|$ cambia para cada figura convexa procesada; resulta conveniente dejar el valor de k dinámico, pues figuras con más puntos tendrán una posibilidad mayor de tener *outliers* y requieren más ciclos de procesamiento, mientras que, de manera inversa, figuras con pocas entradas requerirán pocos ciclos. Dado que se tiene una probabilidad muy pequeña de obtener *outliers* e, se puede considerar $q = \frac{|s|}{2}$. Finalmente, sustituyendo los valores de q, e y Pr en la Ecuación 6.1 y tomando la cota superior para k, se tiene:

$$k \approx \left\lceil \frac{-1.30102}{log(1-(0.95)^{\frac{|s|}{2}})} \right\rceil \tag{6.2}$$

6.1.2 Criterio de paro b

El criterio de paro b puede ser establecido a partir de la probabilidad de obtener un *outlier* y de la cardinalidad del conjunto de entrada s, de forma que el criterio de

paro cumpla con la probabilidad de obtener únicamente *inliers*, como se muestra en la Ecuación 6.3:

$$b = (1 - e)\,|s| \tag{6.3}$$

Siendo e un valor conocido, es posible sustituirlo en la Ecuación 6.3 y redondear al piso, pues se busca un número mínimo de valores que ajustan a m. Así, la Ecuación 6.4 representa nuestro criterio de paro:

$$b = \lfloor 0.95\,|s| \rfloor \tag{6.4}$$

Finalmente, el parámetro t depende del rango de valores del conjunto de entrada s y del porcentaje de error tolerable. En nuestro caso, este porcentaje es determinado por la caracterización del sensor. El mismo trabajo de [Choo et al., 2014] menciona que la desviación del sensor es de $\pm 3\,\%$, mientras que en los datos técnicos de Kinect se menciona que el tamaño nativo de la cámara RGB es de 1280 x 960 pixeles, lo que da como resultado un umbral $t \approx 38$, que es la máxima desviación que tendría el sensor.

6.2 Langragiano aumentado

En la Sección 6.3, se muestra cómo expresar una primitiva tridimensional, mediante sus ecuaciones paramétricas, e integrarla como la función objetivo de un problema de optimización con restricciones. Teniendo la libertad de modelar el problema, lo expresaremos únicamente con restricciones de igualdad, de manera que su resolución sea lo más sencilla posible. Para el proceso de optimización que ajusta la nube de puntos de entrada S a un modelo de primitiva tridimensional, usamos el Langragiano aumentado y modificado, propuesto por Shtof [Shtof et al., 2013]. Para explicarlo, primeramente es necesario recordar que un problema de optimización, sujeto a ecuaciones de igualdad, puede ser expresado como se muestra en la Ecuación 6.5:

$$\begin{aligned} &\min && f(x) \\ &\text{subjeto a} && g_i(x) = 0,\ i = 1, \ldots, |G|\,. \end{aligned} \tag{6.5}$$

El Langragiano aumentado, como muchos otros métodos, resuelve un problema con restricciones separándolo en varios problemas sin restricciones. Para ello, se define una constante μ y un conjunto de constantes $\lambda = \{\lambda_1, \lambda_2 \ldots \lambda_{|G|}\}$. Por lo tanto, definimos

el Langragiano aumentado mediante la función objetivo sin restricciones, mostrada en la Ecuación 6.6:

$$L_{aug}(x, \mu, \lambda) = f(x) - \sum_{i=1}^{|G|} \lambda_i \cdot c_i(x) + \frac{\mu}{2} \sum_{i=1}^{|G|} |c_i(x)|^2 \tag{6.6}$$

En la Ecuación 6.6 el primer término corresponde a la función objetivo original, el segundo es el término del Langragiano y el tercero es el término de penalización. De forma intuitiva, se puede observar que, al minimizar x, se intenta aproximar las condiciones de KKT [Nocedal and Wright, 2006], como se muestra en la Ecuación 6.7:

$$\begin{aligned} \nabla_x L &= 0 \\ c_i(x) &= 0 \end{aligned} \tag{6.7}$$

El algoritmo del Langragiano aumentado se basa en un ciclo iterativo que minimiza la función objetivo sin restricciones, con un ciclo anidado que actualiza los parámetros λ para tener una mejor aproximación de los multiplicadores de Lagrange y, por lo tanto, las condiciones de KKT. Una actualización tan simple es posible, ya que el gradiente de L_{aug} y el de la función de Lagrange son muy similares cuando x se encuentra cerca de satisfacer las restricciones, lo cual se ve claramente en el Algoritmo 9.

Input: $x^{(0)}$, $\lambda^{(0)}$, μ
i=0;
while *no converge* **do**
  $x^{(i+1)} = \underset{y}{\text{argmin}} L_{aug}(y, \lambda^{(i)}, \mu)$; `// comenzando desde` $x^{(i)}$
  for *j=1 to* $|G|$ **do**
    $\lambda_j^{(i+1)} = \lambda_j + \mu \cdot c_j(x^{(i+1)})$;
  end
  i++;
end

Algoritmo 9: Langragiano aumentado

Para acelerar la convergencia del Langragiano aumentado, [Shtof et al., 2013] propone una actualización del factor μ a partir de un factor μ_0. La lógica detrás del proceso de aceleración es que si la violación a las restricciones no se reduce lo suficiente, se incrementará el valor de μ, como se puede observar en el Algoritmo 10. Dado que este procedimiento requiere de valores iniciales para una correcta convergencia, el mismo autor ofrece los valores de inicialización más adecuados con base en experimentación. De esta manera, se usan $\lambda_0 = 10$, $\lambda_{max} = 1000$, $\alpha = 0.5$, $\beta = 2$ y $\lambda_i^0 = 0$.

Input: $x^{(0)}$, $\lambda^{(0)}$, μ_0,μ_{max},α,β
i=0;
$\mu = \mu_0$;
$C_{max} = \frac{1}{\mu^\alpha}$;
while *no converge* **do**
  $x^{(i+1)} = \underset{y}{\operatorname{argmin}} L_{aug}(y, \lambda^{(i)}\mu)$;
  ; `// comenzando desde` $x^{(i)}$
  for *j=1 to* $|G|$ **do**
    $\lambda_j^{(i+1)} = \lambda_j + \mu \cdot c_j(x^{(i+1)})$;
  end
  if $\sum_{i=1}^{|G|} \left|c_i(x^{(i+1)})\right|^2 < C_{max}$ **then**
    $C_{max} = \frac{C_{max}}{\mu^\alpha}$;
  end
  else
    $\mu = \min(\mu \cdot \beta, \mu_{max})$;
    $C_{max} = \frac{1}{\mu^\alpha}$;
  end
  i++;
end

Algoritmo 10: Langragiano aumentado y acelerado

Podría pensarse que un método de penalización con un parámetro de entrada μ podría resolver también la función objetivo con restricciones de igualdad, de una manera más sencilla, asumiendo la forma de la Ecuación 6.8:

$$L_{pen}(x, \mu) = f(x) + \frac{\mu}{2} \cdot \sum_{i=1}^{|G|} |c_i(x)|^2 \tag{6.8}$$

Teóricamente, la Ecuación 6.8 converge a la solución que cumple con las restricciones cuando $\mu \to \infty$. Sin embargo, el problema es que, en la práctica, no se pudo llegar a una solución que pudiera actualizar los valores de μ y que converja rápidamente, sin estancarse en mínimos locales debido a la distorsión de las formas. Así, el Algoritmo 10 es resistente a la no convexidad de la solución.

6.3 Primitivas tridimensionales

Desde el punto de vista del diseño y la manufactura, los objetos de uso diario encuentran eficiencia y belleza al formarse de primitivas tridimensionales. Los cortes necesarios para formar las piezas que los conforman, al ser lineales o con curvas estables, hacen más sencilla la manufactura y reducen costos. Por esta razón, basta con ver alrededor para darse cuenta que es posible reducir la mayoría de los objetos, que nos rodean en la vida diaria, a estructuras geométricamente simples, exceptuando quizá a objetos decorativos, cuya función principal es la de representar formas churrigurescas o naturales, lo cual está fuera del alcance de este trabajo.

Entre la gran variedad de posibles primitivas tridimensionales, muchas de ellas aparecen únicamente en un pequeño porcentaje de los objetos. El cubo, el prisma rectangular o caja, la esfera, el cilindro y el cono son las primitivas más recurrentes en la formación de objetos hechos por el ser humano, mientras que primitivas como pirámides, toroides, prismas hexagonales, entre otras, no son recurrentes en el diseño industrial o tienen muy poca aparición [Li et al., 2011], [Figueiredo et al., 2017]. Además, la identificación de un mayor número de primitivas tendrá resultados directos en la complejidad del algoritmo propuesto, como se muestra más adelante en la Sección **??**. Incluso, al elevar el número de objetos clasificables, no deberían considerarse más primitivas tridimensionales, a menos que estas aparezcan de manera recurrente en nuevos objetos.

Como se ha visto, el algoritmo RANSAC necesita como entrada una función de evaluación con la cual el error será minimizado, por lo que las primitivas tridimensionales deben ser representadas de una forma conveniente, i.e., como una función objetivo. De hecho, el trabajo de [Shtof et al., 2013] ya propone las funciones objetivo de las primitivas tridimensionales más importantes. Si bien este trabajo aplica estas funciones usando sólo el método de Langragiano aumentado, en nuestro caso, las adaptaremos

para usarlas con RANSAC. La justificación para esto es que el método del Langragiano aumentado, en general, es computacionalmente costoso y, además, podemos vernos afectados sobremanera por *outliers*. Por ello, sólo lo usaremos como punto de partida.

6.3.1 Esferas

Es necesario recordar que el objetivo de RANSAC es minimizar un error, por lo que debemos expresar las primitivas tridimensionales como funciones objetivo que deben ser minimizadas, mas no como ecuaciones directas. Para entender este concepto, considere la función objetivo más sencilla, la de la esfera (ver Ecuación 6.9):

$$m_{esfera} = \sum_{i=1}^{N} [\| C_{x,y} - s_i \|^2 - r^2]^2 \tag{6.9}$$

De la Figura 6.3 y la Ecuación 6.9, se puede deducir que r es el radio de la esfera y que s_i es el i-ésimo punto de la superficie convexa sujeta a análisis, mientras que $C_{x,y}$ representa el par ordenado de las coordenadas del centro de la esfera. Para encontrar los primeros valores de r y $C_{x,y}$, que conforman el primer modelo m del algoritmo RANSAC, simplemente se aplica el método de Langragiano aumentado a la función objetivo definida en la Ecuación 6.9, usando el conjunto inicial que se ha escogido en el algoritmo RANSAC. Para el resto de los puntos del conjunto de entrada, se continua aplicando el algoritmo RANSAC, como se mencionó en la Sección 6.1, i.e., ajustaremos el resto de los puntos al modelo de la esfera que hemos obtenido, midiendo su error err y repitiendo este proceso un numero k de veces para determinar el modelo m que minimiza err.

Tanto para la esfera como para las demás figuras tridimensionales, en este trabajo se utiliza, como medida de error, el número de *inliers*, i.e., $\frac{1}{|posiblesInliers|}$. El caso de la esfera es perfecto para explicar cómo funciona nuestra versión de RANSAC: una vez que se ha creado el modelo, mediante el método del Langragiano aumentado, es necesario ajustar los puntos sobrantes al modelo. Para ello, se debe usar una medida de distancia, que en este caso es sencilla como se observa en la Ecuación 6.10, pues basta con obtener la diferencia entre el valor del radio r del modelo y la distancia euclidiana entre el centro $C_{x,y}$ y el punto s_i:

$$err_{esfera} = \|r - Distance(C_{x,y}, S_i)\| \tag{6.10}$$

La Ecuación 6.10 sustituye a la distancia $\|s_i - m(s_i)\|$ en el algoritmo 8 para términos de referencia, esta distancia será denominada como la distancia del modelo al punto.

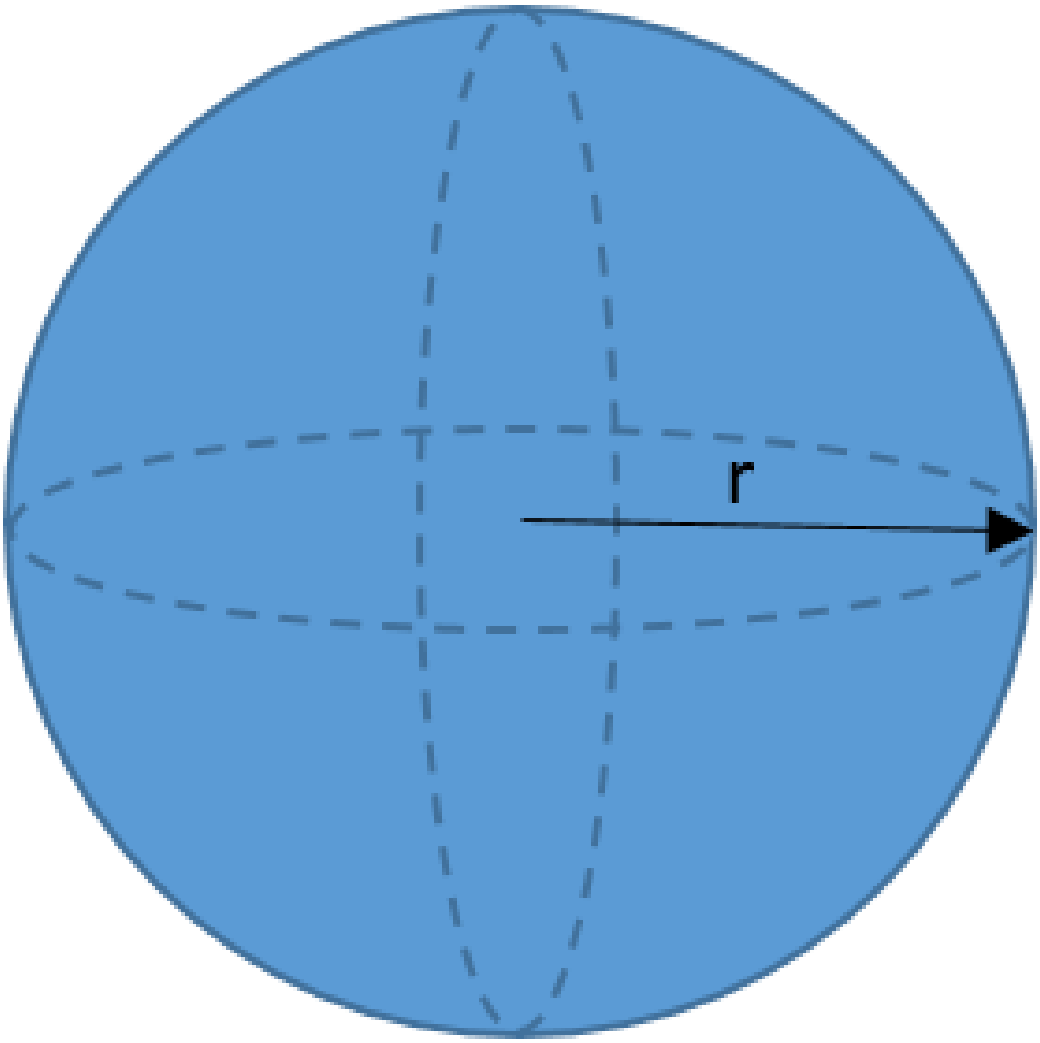


Figura 6.3: Modelo de la esfera para ser representada como una función objetivo

Si se analiza el caso de la esfera, es fácil observar que son pocos los cambios a realizar para el resto de las primitivas. Sólo se debe encontrar un modelo m por el método del Langragiano aumentado, a partir de la propuesta de Shtof [Shtof et al., 2013] y una forma de medir la distancia de un punto s_i al modelo m, lo cual será inherente a todas las primitivas tridimensionales que se desee analizar.

El caso de la esfera resulta práctico para entender el problema pero, a partir de este punto, el análisis de las primitivas tridimensionales comienza a complicarse. Por ejemplo, en el caso del cilindro, lo más natural sería pensar que se puede obtener una medida de error similar a la de la esfera, donde simplemente se calcule la distancia al centro del plano circular. Sin embargo, esta idea resulta inadecuada cuando el cilindro se encuentra rotado en su eje principal, trasladado a un punto distinto del origen y escalado en sus proporciones. En el caso de la esfera no había problema, ya que su simetría la vuelve resistente a todos estos cambios geométricos, pero las demás primitivas si se ven afectadas.

6.3.2 Conos y cilindros

En el caso del cono, este se encuentra limitado por dos superficies circulares paralelas entre sí. De hecho, es posible definir el cuerpo del cono a partir de estas superficies. Así, se puede definir una primera función objetivo dedicada a encontrar las superficies circulares, como se muestra en la Ecuación 6.11. De esta, se puede observar que se comporta de una forma muy similar a la esfera, pues se busca adaptar los puntos s_i a una superficie circular con un radio r y un centro c_{xy}, pero con la diferencia de que para ubicarlo en el espacio tridimensional, se utilizará un vector normal a la superficie n_{xyz}:

$$\phi_{circle}(c, n, r) = \sum_{i=1}^{N} [\| \, n_z(c_{xy} - s_i) \, \|^2 + (n_{xy} \cdot (c_{xy} - s_i))^2 - (rn_z)^2]^2 \tag{6.11}$$

Una vez encontradas las tapas circulares, la definición del volumen cónico es muy sencilla. Sólo es necesario definir una función objetivo, a partir de las tapas circulares, que relacione el largo del cono, como se muestra en la Ecuación 6.12, en la que se relacionan ambas tapas haciendo uso del largo l:

$$\phi_{cone} = \frac{1}{2}\phi_{circle}(c, n_{bottom}, r_{bottom}) + \frac{1}{2}\phi_{circle}(c + ln_{top}, n_{top}, r_{top}) \tag{6.12}$$

En la Figura 6.5, se observa que el trabajo de Shtof [Shtof et al., 2013] considera que el vector normal n_{xyz} es idéntico en ambos círculos. Esta suposición funciona bien para este trabajo, pues se construye el *s*ketch de formas ya establecidas, pero al

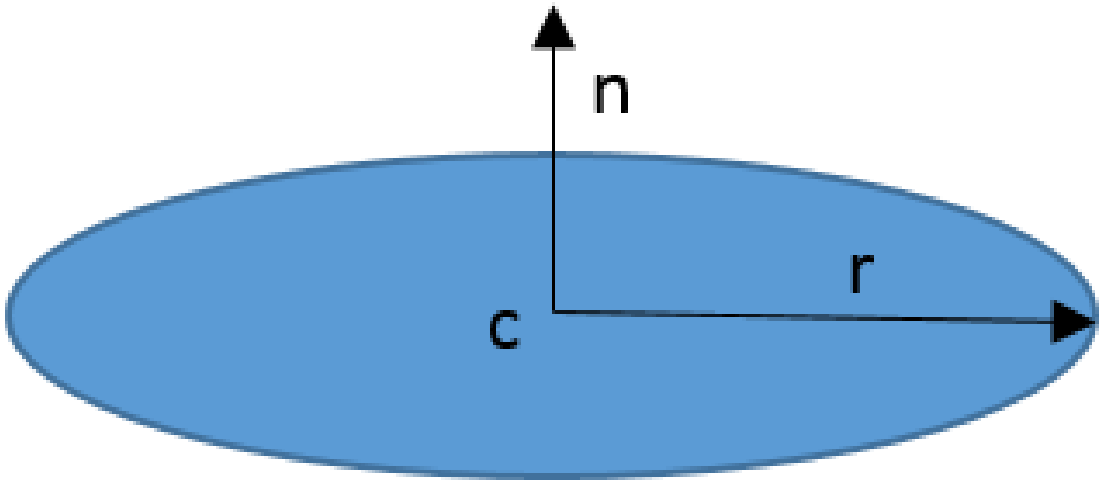


Figura 6.4: Base circular y sus variables como función auxiliar para minimizar el error en conos y cilindros

tomar objetos del mundo real no se puede dar por hecho de que esto será así, por lo que para el presente trabajo se establece una restricción $n_{bottom} \cdot n_{top} = 1$, lo que se traduce de la siguiente manera: los vectores normales de la tapa inferior n_{bottom} y de la tapa superior n_{top} deben ser paralelos entre sí. La Ecuación 6.12 funciona tanto para conos como para cilindros, ya que un cilindro será un cono cuyas dos tapas tienen un radio tal que $|r_{bottom} - r_{top}| < \omega$ para un valor de ω lo suficientemente pequeño.

El cálculo del error para RANSAC por cada punto resulta un poco más complicado. Como ya se ha mencionado el cilindro puede estar rotado, trasladado y escalado, por lo que el cálculo de la distancia entre el punto y la superficie cónica se complica, siendo matemática y computacionalmente complejo. Una manera más sencilla de realizar el cálculo del error es pasar el modelo ϕ_{cone} a un espacio donde no tiene rotaciones y se encuentra posicionado en el origen. Para ello, sólo tenemos que obtener la matriz de transformación RT definida en la Ecuación 6.13:

$$RT = \begin{bmatrix} R_{1,1} & R_{1,2} & R_{1,3} & T_x \\ R_{2,1} & R_{2,2} & R_{2,3} & T_y \\ R_{3,1} & R_{3,2} & R_{3,3} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.13}$$

De la Ecuación 6.13, los factores de traslación se pueden obtener fácilmente si ponemos el centro c_{xyz} de la base del cono hacia el origen, siendo $T_x = -c_x$, $T_y = -c_y$ y $T_z = -c_z$. Los factores de R se obtienen al multiplicar las tres matrices de rotación, tomando en cuenta los ángulos α, β y γ que representan la rotación en los ejes x, y

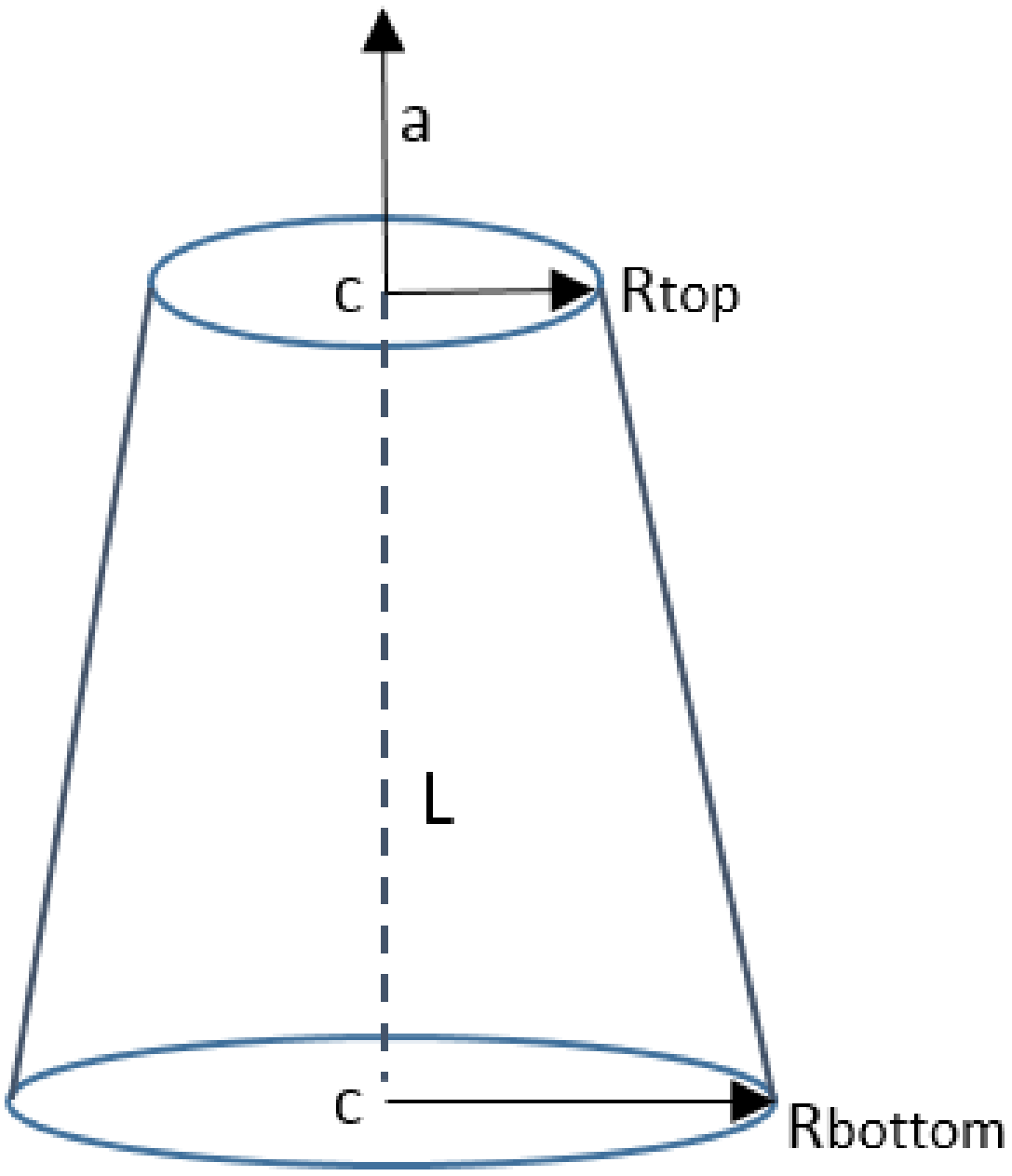


Figura 6.5: Variables a ajustar en el modelo del cono y del cilindro

y z, respectivamente, y que pueden ser deducidos fácilmente a partir de cualquiera de los dos vectores normales de las tapas del cono, tomando el sentido negativo de la rotación del vector normal. Esto se expresa mejor en la Ecuación 6.14:

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha \\ 0 & sin\alpha & cos\alpha \end{bmatrix} \begin{bmatrix} cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -sin\beta & 0 & cos\beta \end{bmatrix} \begin{bmatrix} cos\gamma & -sin\gamma & 0 \\ sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.14}$$

La matriz RT resulta muy útil, pues en una sola operación matricial, se pasa un punto s_i a un espacio donde se puede determinar la distancia al cono; por mucho, esto es más sencillo que en su forma original. Para ello, sólo debemos proceder mediante la Ecuación 6.15, obteniendo un punto p':

$$p' = \begin{bmatrix} R_{1,1} & R_{1,2} & R_{1,3} & T_x \\ R_{2,1} & R_{2,2} & R_{2,3} & T_y \\ R_{3,1} & R_{3,2} & R_{3,3} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_{i_x} \\ s_{i_y} \\ s_{i_z} \\ 1 \end{bmatrix} \tag{6.15}$$

En este nuevo espacio, se puede obtener la distancia del punto p' al cono con una sencilla modificación de la distancia Manhattan [Black, 1998] (ver Ecuación 6.16), que consiste en obtener la distancia al centro del cono y restarle el valor del radio dominante en esa posición de la vertical. El desarrollo de esta modificación es el siguiente, supóngase que C es el valor del centro de la tapa superior en el nuevo espacio (recordando que siempre se busca que las rotaciones coincidan para que $r_{bottom} < r_{top}$) entonces se cumple que los puntos s_i, tales que $0 \leq s_{i_y} \leq C_y$, tienen que calcular su distancia a un radio dinámico, pues este crece desde r_{top} hasta r_{bottom}; los puntos, tales que $s_i > C_y$, tienen que calcular su distancia hacia r_{top}; y los puntos, tales que $s_i < 0$, deben calcular su distancia hacia r_{bottom}; todas estas condiciones están representadas por la variable Ψ_1. La ecuación resultante es sólo la suma de los valores absolutos de las coordenadas de s_i hacia la línea imaginaria que conecta los centros de ambas tapas y después la resta de la distancia hacia el valor del radio en esa coordenada de y, lo que da como resultado la distancia de s_i al cuerpo del cono; en caso de no estar alineado en y, se debe sumar también el valor absoluto de la coordenada y de s_i al cálculo de la distancia; estas condiciones están representadas por la variable Ψ_2. De esta manera, el cálculo de la distancia se ve reflejado, de manera sencilla en la Ecuación 6.16.

$$err_{cono} = |x| + |y| + \Psi_1 + \Psi_2 \tag{6.16}$$

Las condicionales Ψ_1 y Psi_2 quedan definidas en función del valor de y, según se muestra en las Ecuaciones 6.17 y 6.18, por lo que es necesario recalcar el cuidado que

se debe tener en buscar que la matriz de rotación R haga coincidir el eje principal del cono con el eje y.

$$\Psi_1 = \begin{cases} r_{bottom}, & \text{if } s_{i_y} < 0 \\ r_{top}, & \text{if } s_{i_y} > C_y \\ -(r_{top} + (r_{top} - r_{bottom})\frac{C_y - s_{i_y}}{C_y}), & \text{otherwise} \end{cases} \tag{6.17}$$

$$\Psi_2 = \begin{cases} \left|s_{i_y}\right|, & \text{if } s_{i_y} < 0 \\ s_{i_y} - C_y, & \text{if } s_{i_y} > C_y \\ 0, & \text{otherwise} \end{cases} \tag{6.18}$$

Toda la justificación matemática puede parecer complicada, pero se resume sencillamente en que, para un cono propuesto, se obtiene la matriz RT mediante la Ecuación 6.15. Para todos los puntos se aplica esta matriz y se usa la Ecuación 6.16 para calcular la distancia.

Resulta conveniente resaltar que un cilindro es una particularidad de un cono, con la característica de que sus dos tapas tienen el mismo radio. En general, para este trabajo, se puede ajustar una figura a un cilindro, si se sabe que $|r_{bottom} - r_{top}| < \omega$ para un valor de ω suficientemente pequeño. Se puede hacer el ajuste necesario, agregando la restricción de igualdad $r_{bottom} - r_{top} = 0$, con lo que el valor de Ψ_1 es igual al radio de una de las tapas, en cualquiera de los casos, lo que simplifica el cálculo.

6.3.3 Prismas rectangulares

En el caso del prisma rectangular, la función objetivo propuesta en la Ecuación 6.19 se complica, pues ahora es necesario encontrar más parámetros, restringidos entre ellos. Como se muestra en la Figura 6.6, ahora la ecuación paramétrica toma un punto de referencia que actúa como la esquina inferior izquierda (esto es especialmente conveniente para el uso de Kinect) y a partir de ella se desprenden los parámetros de la ecuación 6.19.

$$\begin{aligned} \Phi_{prism} =& \alpha(h_0 d_0 \parallel W - W_0 \parallel^2 + w_0 d_0 \parallel H - H_0 \parallel^2 + w_0 h_0 \parallel D - D_0 \parallel^2) \\ &+ (w - w_0)^2 + (h - h_0)^2 + (d - d_0)^2 + \parallel c_{xyz} - c_0 \parallel^2 \end{aligned} \tag{6.19}$$

La función objetivo representada en la Ecuación 6.19 se ve sujeta a las restricciones: $\parallel W \parallel^2 = 1$, $\parallel H \parallel^2 = 1$, $\parallel D \parallel^2 = 1$, $W \cdot H = 0$, $W \cdot D = 0$ y $H \cdot D = 0$ para mantener la ortonormalidad del sistema. En la misma ecuación se tiene el parámetro

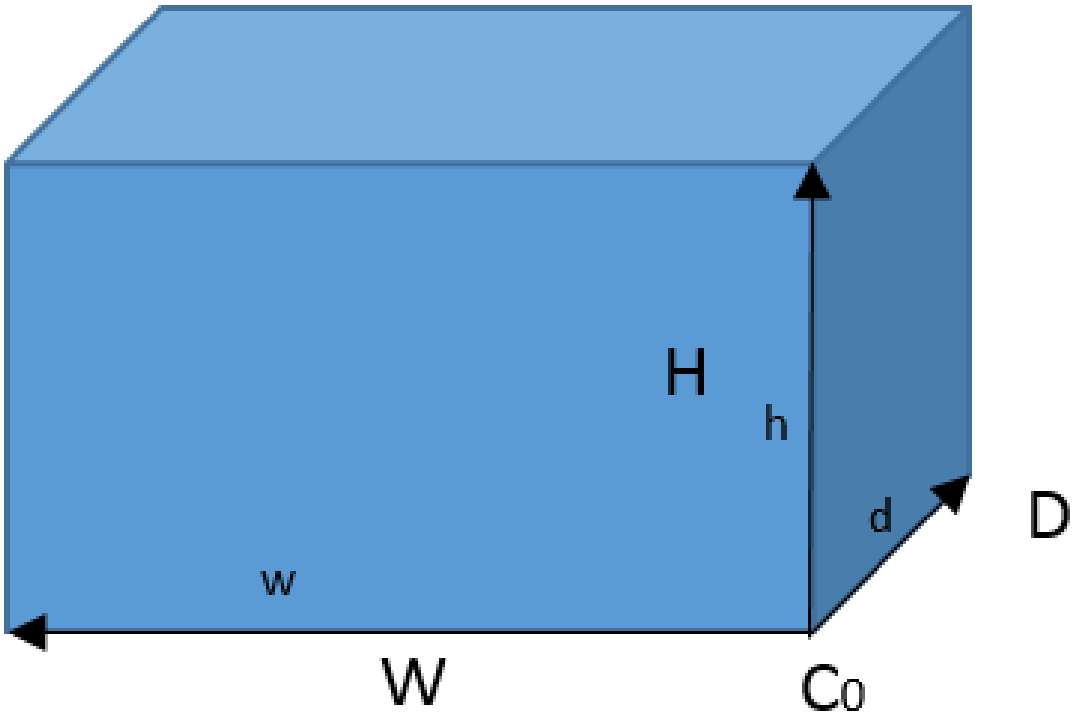


Figura 6.6: Representación gráfica de los parámetros a minimizar en la función objetivo del prisma

α, el cual simplemente es un valor de peso que, en el presente trabajo, fue ajustado experimentalmente a 0.1. Al igual que en las primitivas tridimensionales anteriores, es necesario buscar una manera de generar un primer modelo, que en este caso es la función objetivo de la Ecuación 6.19, así como una forma de medir el error entre un punto y el modelo propuesto. En el caso de un prisma rectangular, se presenta el mismo problema que se tenía con los cilindros, ya que el objeto puede estar rotado, trasladado y escalado en tres dimensiones, por lo cual la misma solución resulta factible. Así, únicamente se tiene que encontrar una matriz de transformación que lleve el prisma al origen sin rotaciones, lo que permitirá calcular de forma sencilla la distancia.

Para construir la matriz RT, se toma en cuenta el punto c de la Ecuación 6.19 que indica la esquina a partir de la cual se construye el prisma. En nuestro caso, dadas las propiedades de Kinect, esta esquina siempre será la frontal inferior izquierda. Primeramente, se tiene que las rotaciones pueden ser obtenidas a partir de la triada de vectores W, H y D, recordando que estos deben ser paralelos a los ejes coordenados. Entonces, basta con calcular la diferencia angular hacia ellos, de forma que se puedan obtener los valores de α, β y γ para construir R usando la Ecuación 6.14. Para obtener el valor de las traslaciones, se usan los valores de las dimensiones w, h y d y de la esquina c, pasando primero el origen hacia la esquina c y después moviéndolo en proporción a las dimensiones para ajustar el origen con el centro del prisma, de forma que el valor de T queda dado por la Ecuación 6.20:

$$T = \left[\frac{w}{2} - c_x, \frac{h}{2} - c_y, \frac{d}{2} - c_z\right] \tag{6.20}$$

Para hacer que la matriz RT sea más sencilla en su manejo, se agrega un factor de escalamiento E, de forma que en el nuevo espacio de transformación el prisma rectangular vaya de -1 a 1 en todos sus ejes. Así E es una matriz que escala a la matriz RT, según las dimensiones w, h y d, como se muestra en la Ecuación 6.21:

$$E = \begin{bmatrix} \frac{2}{w} & 0 & 0 & 0 \\ 0 & \frac{2}{h} & 0 & 0 \\ 0 & 0 & \frac{2}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.21}$$

Al procesar el punto s_i, no se hace uso de la matriz RT como en el caso del cilindro, sino que se emplea la multiplicación del factor de escalamiento E y la matriz RT, a lo que se denomina como matriz ERT. Al igual que en el caso del cilindro, se modifica la distancia Manhattan [Black, 1998] para obtener la distancia desde el punto s_i al cuerpo de la primitiva tridimensional. Es importante notar que, con esta modificación, se busca obtener la distancia a las aristas del prisma. Es aquí donde cobra sentido

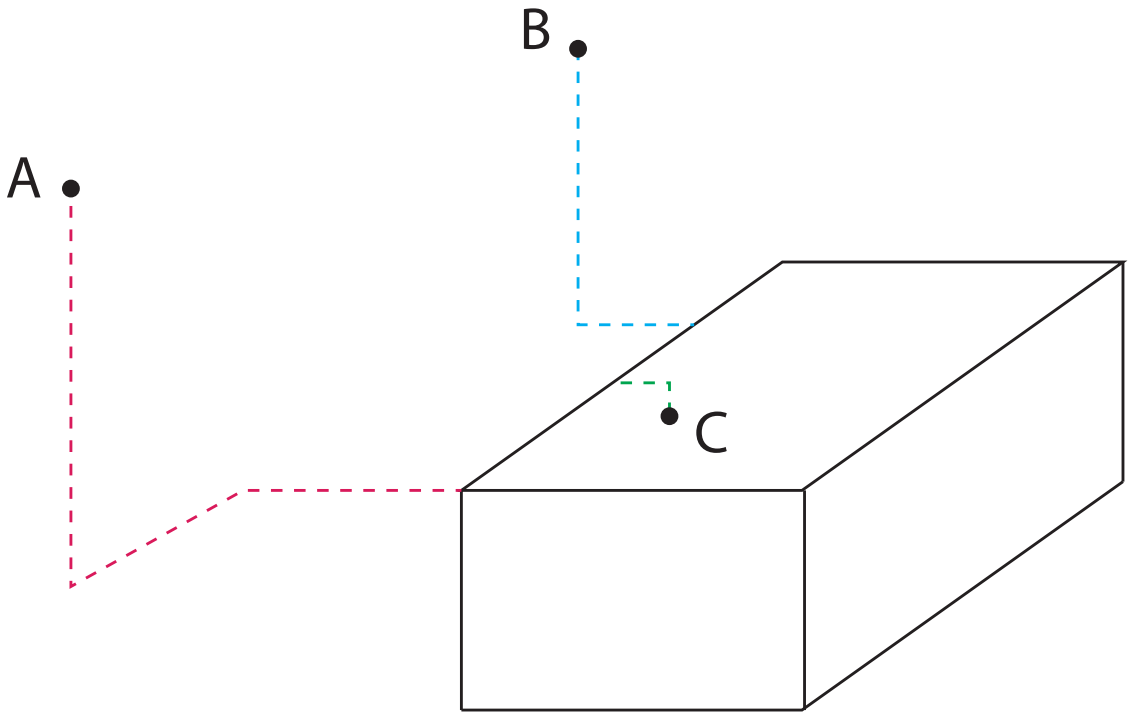


Figura 6.7: En A se observa cómo usando la Ecuación 6.22, un punto alejado en las tres coordenadas se acerca al vértice más cercano. En B es posible observar que si un punto se encuentra alineado en una o dos coordenadas, sólo se aproxima a la arista más cercana, ignorando las coordenadas que ya se encuentran alineadas. En C se muestra que, en el caso donde el punto está contenido dentro del prisma, sólo se acerca en dos direcciones hacia la arista más cercana.

hacer el escalamiento, usando la Ecuación 6.21: al aplicarlo, las ocho esquinas del prisma quedan marcadas por las combinaciones de la unidad y su negativo, i.e., (1,1,1), (-1,1,1), (1,-1,1), (1,1,-1), (-1,-1,1), (1,-1,-1), (-1,1,-1) y (-1,-1,-1), mientras que las líneas entre estos puntos marcan las aristas. Desde el punto de vista de la distancia de un punto a una de las aristas, si el valor absoluto de una coordenada se encuentra entre la unidad y su negativo, este punto se encuentra alineado en esa coordenada con una de las aristas del prisma; de no ser así, se puede aproximar la distancia mínima en esa coordenada, buscando la diferencia con la unidad, i.e., se busca aproximarlo al vértice más cercano, como se ilustra en la Figura 6.7. La afirmación anterior encuentra una explicación matemática sencilla en la Ecuación 6.22, que define la función de distancia para la medición del error entre el punto s_i y el modelo m propuesto.

$$err_{prisma} = \Upsilon_1 + \Upsilon_2 + \Upsilon_3 \tag{6.22}$$

Donde los valores Υ_1, Υ_2 y Υ_3 son condicionales, tales que:

$$\Upsilon_1 = \begin{cases} 0, & \text{if } -1 < s_{i_x} < 1 \\ 1 - |s_{i_x}|, & \text{otherwise} \end{cases}$$

$$\Upsilon_2 = \begin{cases} 0, & \text{if } -1 < s_{i_y} < 1 \\ 1 - |s_{i_y}|, & \text{otherwise} \end{cases}$$

$$\Upsilon_3 = \begin{cases} 0, & \text{if } -1 < s_{i_z} < 1 \\ 1 - |s_{i_z}|, & \text{otherwise} \end{cases}$$

Si el valor de la Ecuación 6.22 es igual a cero, puede indicar dos casos: 1) el punto cae exactamente en el modelo, o 2) el punto se encuentra dentro del prisma rectangular. Para distinguir entre estos casos, cuando se obtenga un cero con la Ecuación 6.22, se completará la medición, haciendo uso de la Ecuación 6.23:

$$err_1 = (1 - |x|) + (1 - |y|) + (1 - |z|) - max(1 - |x|, 1 - |y|, 1 - |z|) \tag{6.23}$$

La Ecuación 6.23 acerca el punto a su arista más cercana. El término máximo utilizado es necesario para eliminar la coordenada más lejana, evitando que se acerque de forma innecesaria a un vértice, en lugar de una arista. Además, esta ecuación es complementaria a un caso en específico, por lo que se espera que este caso se alcance en muy pocas ocasiones. El único caso donde las Ecuaciones 6.22 y 6.23 dan cero, de manera simultánea, es cuando efectivamente el punto analizado encaja a la perfección como una arista o vértice del modelo.

6.4 Selección de primitivas tridimensionales

Al utilizar RANSAC de forma complementaria y no únicamente el Langragiano aumentado, es posible tener la medición del error err, a través de una distancia entre la nube de puntos de entrada S y el modelo m propuesto. Esta medición del error es acumulativa y tiene un rango $[0, \infty)$, donde un valor de cero indicaría un ajuste perfecto de S a m. Entre mayor sea el valor de err, los puntos se encuentran más alejados del modelo m. Bajo la premisa anterior, es posible buscar un modelo que minimice err; incluso es posible hacer una búsqueda iterativa selectiva, en la que bajo la acumulación de cierta cantidad de error se deje de buscar que el modelo ajuste a una primitiva. Para ello, se toma como base el Algoritmo 8, sólo que, en lugar de buscar

un solo modelo, se hará una búsqueda de los modelos de todas las primitivas tridimensionales que se deseen. Por adelantado, se puede notar que esta idea es escalable, pues si bien este trabajo sólo emplea esferas, conos, cilindros y prismas rectangulares, la metodología es la misma aún si se agregan o restan primitivas.

El Algoritmo 11 muestra el proceso principal para ajustar primitivas tridimensionales de un modo no supervisado, mientras que el Algoritmo 12 presenta la modificación hecha con base en el Algoritmo 8, que permite parar el proceso, dada la condición de que el error acumulado no sobrepase una variable Γ y que además recibe, como entrada, un método para calcular el modelo m, usando las Ecuaciones 6.9, 6.12 y 6.19, así como un método para calcular el error entre la nube de puntos y el modelo dado; estos últimos se basan en las Ecuaciones 6.10, 6.16 y 6.22.

Input: s, n, k, t, b,Γ
$[Error_{Esfera}, M_{Esfera}]$=RansacAjuste($s$, n, k, t, b, Γ,$m_{Esfera}()$, $err_{Esfera}()$);
$[Error_{Prisma}, M_{Prisma}]$=RansacAjuste($s$, n, k, t, b, Γ,$m_{Prisma}()$, $err_{Prisma}()$);
$[Error_{Cono}, M_{Cono}]$=RansacAjuste(s, n, k, t, b, Γ,$m_{Cono}()$, $err_{Cono}()$);
minError=minimo($Error_{Esfera}$, $Error_{Cono}$, $Error_{Prisma}$);
switch ($minError$)
case $Error_{Esfera}$**:**
M_{final}=m_{Esfera};
case $Error_{Cono}$**:**
M_{final}=m_{Cono};
case $Error_{Prisma}$**:**
M_{final}=m_{Prisma};
end switch
return M_{final};

Algoritmo 11: Ajuste no supervisado de primitivas tridimensionales a un conjunto de puntos de entrada S

Cabe resaltar que, en la medición del error, se debe aplicar las transformaciones geométricas propuestas para cada primitiva tridimensional, pero sólo se calcula una vez cada vez que hay un nuevo modelo, i.e., una vez por iteración, lo que hace sencillo su cálculo en el resto de las iteraciones. Así mismo debe notarse que, al no ser información dependiente, el Algoritmo 11 puede ser ejecutado de forma paralela para

todas las primitivas tridimensionales.

```
Input: s, n, k, t, b, Γ, Modelo(), MedidaError()
for i=1 to k do
    InliersIniciales=SeleccionarAleatoriamente(s, n);
    acumulado=0;
    if acumulado < Γ then
        m=Modelo(InliersIniciales);
        ac=0;
        posiblesInliers=NULL;
        foreach s_i ∈ s | s_i ∉ InliersIniciales do
            e=MedidaError(m,s_i);
            ac+=e;
            if e < t then
                posiblesInliers.Add(s_i);
            end
        end
        if |posiblesInliers| ≥ b then
            if ac < Error then
                M = m;
                Error = ac;
            end
        end
        acumulado+=ac;
    end
    else
        ;
    end
end
return [Error, M];
```

Algoritmo 12: Procedimiento RansacAjuste. RANSAC modificado para hacer el ajuste de una primitiva tridimensional, dado un modelo y un error que entran como parámetros iniciales

Los valores de n, k, t y b, ya han sido establecidos en la Sección 6.1 y en las Subsecciones 6.1.1 y 6.1.2, mientras que la rutina para cada modelo corresponde a las Ecuaciones 6.9, 6.12 y 6.19. Por su parte, los procedimientos para calcular el error de una primitiva tridimensional, dado un conjunto de entrada s, se encuentran expresados en las Ecuaciones 6.10, 6.16 y 6.22. Queda entonces determinar el valor de Γ, que puede ser obtenido a partir del valor del error medido del sensor. Como se mencionó en la Subsección 6.1.1, el trabajo de Pluskal [Pluskal et al., 2010] maneja un error de 5 % para Kinect, por lo que el error máximo esperado para un modelo se establece como la suma del máximo error de cada uno de los puntos, tal como se establece en

la Ecuación 6.24:

$$\Gamma = 0.05 * (1 + t) * |s| * k \tag{6.24}$$

La Ecuación 6.24 expresa el máximo error esperado antes de que una primitiva tridimensional sea rechazada, este error se describe como el 5 % de la máxima distancia entre un punto de s y una primitiva tridimensional $(1+t)$ multiplicado por el número de puntos de entrada ($|s|$) por el número de iteraciones (k). Una primitiva tridimensional que se ajusta a los puntos podrá fácilmente iterar sin superar el valor de *Gamma*, mientras que una primitiva tridimensional que no encaja al conjunto de puntos, rápidamente superará el valor de Γ.

Escogiendo correctamente el criterio de paro, se puede demostrar que la complejidad computacional de RANSAC es de $O(k \cdot s)$ [Ask et al., 2014]. Es importante configurar el criterio de paro ya que, de modo exhaustivo, RANSAC puede alcanzar hasta $O(s^3)$. Esta complejidad se eleva por el número de primitivas computacionales deseadas, el cual será denotado como Q, y por el número de primitivas que forman un objeto, el cual será representado por F. Por lo tanto, el Algoritmo 11 tiene una complejidad de $O(F \cdot Q \cdot k \cdot s)$.

Al limitar el número de primitivas tridimensionales, de forma que $Q << s$ (en nuestro caso $Q = 3$) se pensaría que este término es despreciable. De la misma manera, esto aplica para el termino F pues, como se menciona en la Sección 7.1.4, resulta conveniente representar un objeto complejo con a penas ocho primitivas tridimensionales o menos, por lo tanto $F << s$. En general, el tamaño máximo de F es 8 y el tamaño contante de Q es 3. En la Sección 7.1.4 se establece una búsqueda de dos esferas, dos cilindros y cuatro prismas. Conociendo el número de parámetros de las Ecuaciones 6.9, 6.12 y 6.19 y multiplicando por el número de primitivas, se puede establecer el número máximo de vértices procesables en el segundo orden de magnitud. Al escoger adecuadamente k, es posible lograr que $F \cdot Q \cdot k$ se encuentre en el mismo orden de magnitud que s, de forma que $F \cdot Q \cdot k \approx s$, buscando una aproximación cuadrática respecto al número de vértices procesados.

6.5 Restricciones geosemánticas

El procedimiento descrito hasta el momento podría parecer suficiente, hablando de un proceso de ajuste de primitivas tridimensionales. En realidad, si sólo se estuvieran analizando primitivas individuales, el proceso podría parar con los resultados presentados en la Sección 6.4. Sin embargo, se debe recordar que se están analizando objetos hechos por el ser humano, razón por la que las primitivas tridimensionales se encuentran relacionadas entre ellas, bajo principios geométricos y semánticos, respondiendo a ángulos y formas con los que los objetos funcionales son diseñados. La

Sección 6.5.1 describe las relaciones geométricas y semánticas más comunes según diferentes autores, así como la aplicación que tienen en los diferentes objetos hechos por el ser humano. Para que estas relaciones tengan una repercusión en las ecuaciones que han sido propuestas en este capítulo, será necesario expresarlas como restricciones de las funciones objetivo de cada primitiva tridimensional, lo cual será tratado en la Sección 6.5.2.

6.5.1 Descripción y aplicación de las relaciones geosemánticas

Cuando se habla de primitivas tridimensionales que dependen unas de otras o tienen una forma establecida por parámetros de diseño, entonces es necesario establecer relaciones geométricas y simétricas entre ellas que permitan describir el escenario tridimensional como un todo armónico, no como partes independientes. De hecho, es común encontrar relaciones geométricas en trabajos de ajuste de primitivas tridimensionales [Gal et al., 2009], [Li et al., 2011], [Xu et al., 2011], sobre todo en reconstrucción de escenarios u objetos. Sin embargo, se habla de relaciones geosemánticas cuando, además de similitudes geométricas, se busca que estas relaciones cumplan con un contexto referente al objeto de análisis. El presente trabajo se centra en objetos hechos por el ser humano, por lo que las relaciones que se buscan deben corresponder a un diseño humano [Chen et al., 2016], con ángulos bien marcados y líneas definidas. Las siguientes relaciones fueron establecidas por [Shtof et al., 2013] para procesos de ajuste basados en *sketching*, pero son relaciones geosemánticas aplicables a cualquier proceso de ajuste o reconstrucción de objetos hechos por el ser humano. Es posible incluir otras relaciones que permitan simetrías, equidistancias o ángulos de 45°, pero establecerlas requiere un minucioso estudio de la aparición de estas en los objetos que serán clasificados y una adecuada representación matemática que podría comprometer el costo computacional de nuestra propuesta. Por esta razón, la adición de nuevas primitivas tridimensionales se dejará como trabajo a futuro.

Paralelismo. Sean dos superficies con normales n_1 y n_2 respectivamente. Se dice que estas superficies son paralelas entre ellas, si estas forman un ángulo cero. Expresado en términos matemáticos, esto es $n_1 \times n_2 = 0$. Cabe notar que se habla de superficies planas con una normal única, por lo que esta relación sólo puede ser medida en la tapa de los cilindros, así como en las caras de los prismas rectangulares.

Perpendicularidad. Dos superficies se consideran perpendiculares cuando sus normales n_1 y n_2 forman un ángulo de 90 grados. En términos de vectores, esto se puede expresar como $n_1 \cdot n_2 = 0$. De manera similar a lo que sucede con el paralelismo, las superficies son planas, por lo que las esferas quedan excluidas de esta relación, aplicándose sólo a conos, cilindros y prismas rectangulares.

Colinealidad. Para i puntos que representan el centro de una primitiva tridimensional, donde $i \geq 3$, las primitivas tridimensionales estarán alineadas por sus centros si

se puede dibujar una línea imaginaria que pase por los tres puntos. De forma vectorial, la construcción de esta línea puede ser expresada como $(c_1 - c_i) \times (c_1 - c_{i+1}) = 0$ para todos los valores de $i > 1$. En general, únicamente se comparan los valores de tres primitivas tridimensionales cercanas, por lo que la relación de colinealidad puede expresarse como $(c_1 - c_2) \times (c_1 - c_3) = 0$. Cabe resaltar que, en este caso, todas las primitivas tridimensionales son objetos de colinealidad, aunque en objetos como conos y prismas se pueden usar los centros de las tapas, mientras que en prismas rectangulares se pueden emplear los centros de las caras, ya que el centro de las primitivas tridimensionales pasará por estos puntos.

Concentricidad. Dos centros c_1 y c_2 de dos primitivas tridimensionales son iguales, si la distancia entre ellos tiende a cero, esto es $c_1 = c_2$. Chen et al. [Chen et al., 2013] describe esta propiedad como útil para eliminar trazos o superficies repetidas, pero dada la algoritmia del Capitulo 5, no será necesario aplicarla.

Coplanaridad. Como su nombre lo indica, esta relación presenta el caso de dos o más primitivas que se encuentran alineadas a un mismo plano, i.e., las normales de sus superficies son paralelas y el vector formado entre sus centros es perpendicular a cualquiera de sus normales, lo que se representa como $(c_1 - c_i) \cdot n_1 = (c_1 - c_i) \cdot n_i = 0$. Nuevamente, se habla de superficies planas con un solo vector normal, por lo que la esfera queda fuera de esta relación.

6.5.2 Relaciones geosemánticas como restricciones en la función objetivo

Las relaciones descritas en la subsección 6.5.1 pueden ser expresadas como restricciones de las funciones objetivo 6.9, 6.12 y 6.19. De forma rigurosa se puede decir que cada una de las primitivas p, que ha sido descrita en la Sección 6.3, tiene una serie de parámetros de ajuste, denominados x_p, de manera que el ajuste de una nube de puntos de entrada S, se lleva a cabo utilizando un modelo $m_p(x_p)$, sujeto a una serie de restricciones internas llamadas $C_p(x_p) = 0$. Todos estos elementos fueron tratados en la Sección 6.3, formando un primer sistema de optimización definido según la Ecuación 6.25:

$$\begin{aligned} &\min && m_p(x_p) \\ &\text{sujeto a} && C_p(x_p) = 0 \end{aligned} \tag{6.25}$$

En esta sección, se han definido los conceptos de primitivas tridimensionales dependientes y de relaciones geosemánticas, por lo que a estas definiciones se agregará un conjunto de relaciones geosemánticas G y por cada relación $g \in G$, se define una restricción $\psi_g(x_g) = 0$; aunque resulta obvio, se debe recordar que $x_g \subset \bigcup_p x_p$.

El proceso de segmentación convexa entrega las diferentes partes de un objeto hecho por el ser humano, así el Algoritmo 11 se ejecutaría J veces si J se considera como

el número de primitivas que forman el objeto analizado. Para la j-ésima iteración, con $1 < j \leq J$, se puede definir el conjunto de las primitivas encontradas como $P = \{p_1, ..., p_j\}$ al cual corresponderá un vector $X = \{x_{p1}, ..., x_{pj}\}$ que representa los parámetros de cada primitiva tridimensional ajustada. Cuando se ha ajustado el modelo de la Ecuación 6.25, es necesario formar el conjunto G que debe relacionar el modelo de primitiva tridimensional con el resto de las primitivas que ya han sido obtenidas. Para ello, se debe verificar cuáles de las relaciones de la Sección 6.5.1 pueden ser aplicadas. Si bien se puede dar una descripción general aplicable a grupos que contengan dos o más primitivas, de forma práctica nuestros grupos sólo contendrán dos o tres primitivas, ya que, al ser un proceso iterativo, consideramos que ya se han aplicado las restricciones geosemánticas a las primitivas tridimensionales analizadas en pasos anteriores del Algoritmo 11.

El objeto tridimensional se desglosa de arriba hacia abajo, según lo dicta el proceso de segmentación convexa. Cuando se alcanza un mínimo de dos primitivas tridimensionales, es posible comenzar a formar grupos. Por generalidad, únicamente se formarán grupos de dos primitivas tridimensionales (o de tres en el caso de la colinealidad). Para formar el grupo G, se debe hacer la verificación de cada una de las relaciones geosemánticas por sus valores numéricos e inecuaciones (aunque las restricciones forzarán a una igualdad). A continuación, se describen las condiciones para añadir cada una de las relaciones geosemánticas a G.

Paralelismo. Sean dos superficies con normales n_1 y n_2 respectivamente. Se añade la restricción de paralelismo a G, si entre ellas se forma un ángulo cercano a cero. En la práctica se busca que, en general, este ángulo esté entre -5 y +5 grados, lo cual puede ser fácilmente medido mediante un producto escalar.

Perpendicularidad. Para dos primitivas tridimensionales, se añade la restricción de perpendicularidad, cuando sus normales n_1 y n_2 formen un ángulo cercano a los 90 grados. De manera similar a lo que sucede con el paralelismo, se busca obtener un ángulo entre 85 y 95 grados, verificándolo con el uso del producto escalar.

Colinealidad. En este caso, se hace uso de un grupo de tres primitivas tridimensionales con centros c_1, c_2 y c_3. Se hace una traslación de todos los puntos a un nuevo origen en c_1 y se calcula el ángulo entre los vectores c_2 y c_3 trasladados. Con ayuda del producto escalar se obtiene el ángulo entre ambos y, en caso de ser menor a 5 grados, se anexa la condición de colinealidad.

Concentricidad. Si bien esta propiedad no es utilizada en el presente trabajo, dada la algoritmia del Capítulo 5, se puede dejar clara para futuras mejoras o trabajos que tomen como base esta tesis. Además es una propiedad muy sencilla de calcular, pues si los centros c_1 y c_2 de dos primitivas tridimensionales tienen una distancia menor al 5 % de su tamaño, entonces se anexa la restricción de concentricidad a G.

Coplanaridad. Para poder constatar que dos objetos son coplanares, dados sus centros c_1 y c_2, así como sus normales n_1 y n_2, es posible replantear la ecuación de la coplanaridad de la Sección 6.5.1 como una relación de sus miembros que da un valor,

el cual indica su grado de acoplamiento: $|1 - ((c_1 - c_2) \cdot n_1)/((c_1 - c_2) \cdot n_2)| \leq 0.05$. En este caso, se utiliza el valor de 0.05, dado que se está trabajando con valores normalizados y esto representa el 5 %, como porcentaje de error.

Una vez construidos el grupo de restricciones G y el grupo de primitivas P, se procede a hacer el ajuste del modelo completo, usando el modelo de optimización de la Ecuación 6.26:

$$\begin{aligned} \min \quad & \sum_{p \in P} m_p(x_p) \\ \text{sujeto a} \quad & C_p(x_p) = 0 \quad \forall p \in P \\ & \psi_g(x_g) = 0 \quad \forall g \in G \end{aligned} \tag{6.26}$$

Capítulo 7

Clasificación de objetos hechos por el ser humano

La principal aportación entregada por el proceso descrito en el Capítulo 6 es una lista de primitivas tridimensionales que, gracias a las restricciones geosemánticas, se ajustan para encajar entre ellas. Este resultado cumple con uno de los objetivos de este trabajo de tesis pero, al mismo tiempo, sirve como herramienta auxiliar para el proceso de clasificación de objetos hechos por el ser humano, ya que se hará uso de las características de las primitivas tridimensionales arrojadas por el proceso de reconstrucción, como atributos del vector característico del proceso de clasificación.

Históricamente hablando, el uso de un modelo tridimensional para clasificar objetos hechos por el ser humano es una técnica que, a finales de los noventa y principios del nuevo milenio, adquiría importancia. La mayoría de las técnicas que datan de esa época se basaban en el emparejamiento de modelos tridimensionales, previamente creados, con datos de sensores espaciales, procesos estereoscópicos o valores 2D mapeados al espacio 3D [Ip et al., 2002], [Osada et al., 2001], [Huber et al., 2004]. La asertividad de estos modelos decaía rápidamente conforme se iba aumentando el número de clases identificables, pasando de un 90 % para 4 clases a un 65 % para 10 clases. De hecho, los trabajos de la época no pretenden clasificar más allá de 15 clases por la caída tan abrupta en los resultados de clasificación y por el poder de cómputo necesario. Esta aproximación presentaba otro problema importante: era necesario crear modelos tridimensionales para cada nuevo objeto a clasificar, por lo que el trabajo arduo de un modelador 3D era necesario, como parte de la preparación; además, el modelo tridimensional no siempre encajaba a la perfección con el objeto a ser reconocido, el cual podía tener cientos, miles o millones de vértices, complicando sobremanera su cómputo.

Precisamente a principios del nuevo milenio aparecían descriptores de imágenes como HOG [Dalal and Triggs, 2005], SIFT [Lowe, 1999] y SURF [Bay et al., 2006], que acapararon la atención de la comunidad de clasificación de imágenes, pues permitían obtener resultados de clasificación sobresalientes aún cuando se trataran bancos de

datos con múltiples clases. Después de una gran explotación, por parte de la comunidad científica, era obvio que los descriptores eran propensos al problema de la invariancia [Morel and Yu, 2009], quedando al descubierto sus debilidades fuera del laboratorio.

Al no contar con sensores adecuados, la investigación de clasificación de objetos, utilizando información tridimensional, se enfocaría primero en usos militares y en el reconocimiento de objetos hechos por el hombre a gran escala [Inglada, 2007] y después en la modificación de descriptores bidimensionales para espacios tridimensionales [Scovanner et al., 2007], [Chaouch and Verroust-Blondet, 2007], [Steder et al., 2010], obteniendo resultados aceptables (aproximadamente del 80 %) para bancos de datos de tamaño mediano, pero nuevamente estancándose en valores aún bajos y teniendo como expectativa el crecimiento exponencial del número de clases.

A partir de 2010, aparecería una serie de cámaras RGB-D (cámaras RGB con un sensor de profundidad incorporado) comerciales, de bajo costo y accesibles a todo público, como Microsoft Kinect®. Estas cámaras facilitaron mucho la investigación en reconocimiento y reconstrucción de todo tipo de objetos, pues la información sobre la profundidad asociada a cada píxel, mejoraba las capacidades de reconstrucción y daba nuevos datos a los clasificadores. La capa de profundidad permitía a los investigadores obtener datos, procesarlos y optimizarlos, de una mejor manera, lo que ayudó a superar el bache en el que se encontraba la investigación, logrando en promedio resultados del 80 % al 85 % de asertividad para bancos de datos medianos [Schwarz et al., 2015].

Con estas nuevas capacidades en los sensores y un poder de procesamiento mucho mayor al de principios del presente milenio, surgieron nuevos enfoques que comenzaban a utilizar la información tridimensional junto a los nuevos enfoques de clasificación, como *Recurrent Neural Network* [Bell et al., 2016], [Socher et al., 2012] y *Convolutional Neural Networks* [Rangel et al., 2018], [Ben-Shabat et al., 2018], [Zhou et al., 2014] con resultados promedio del 85 % de asertividad para bancos de datos medianos, pero con tiempos de entrenamiento de varios días de cómputo, debido a la cantidad de información manejada.

Surge un enfoque más automatizado que no sólo toma la información tridimensional para un proceso matemático, sino que usa como base la forma y características del objeto en un espacio tridimensional. Además, este enfoque toma la misma idea del enfoque inicial de finales de los noventa y principios del milenio, pero esta vez construyendo, de manera automática, las características de forma y no extrayéndolas de un modelo separado. De esta manera, se superaban las adversidades presentadas en aquel entonces y se le añadían las ventajas del uso de nuevos sensores y técnicas de extracción automática. Para extraer las características de forma en el espacio, se usan principalmente dos técnicas: 1) por un lado, se tienen aquellas que rellenan el espacio de la región de interés, utilizando una primitiva tridimensional básica como cubos o esferas [Wu et al., 2015], [Arvind et al., 2017], [Qi et al., 2016], que son conocidas como técnicas volumétricas y 2) por el otro lado, se encuentran las técnicas en

las que se clasifica nuestro trabajo, se trata de técnicas de extracción de formas por ajuste de primitivas tridimensionales que utilizan, como información de entrada, los clasificadores de las características de las primitivas que forman un objeto de interés [Wohlkinger and Vincze, 2011], [Song and Xiao, 2014], [Wang et al., 2017].

Siguiendo el enfoque de ajuste de primitivas tridimensionales, se hará uso de los datos obtenidos en el Capitulo 6. Como el lector recordará, las primitivas tridimensionales no se describen por sus puntos extremos, sino por parámetros de sus ecuaciones. Son precisamente estos parámetros los que se utilizarán para construir un vector característico, como se verá en la Sección 7.1. Con estos datos, es posible además formar un banco de datos para probar nuestro algoritmo, siguiendo los lineamientos que han establecido trabajos similares, como se verá en la Sección 7.2. Así mismo aprovechando el estado del arte existente, se puede utilizar la metodología de validación dominante (KFCV con $K = 10$) para comparar nuestros resultados con trabajos similares, como se mostrará en la Sección 7.3. Si bien se hará una comparación con artículos que utilizan la técnica de ajuste de primitivas tridimensionales, es importante comparar con otras técnicas actuales, como CNN (Convolutional Neural Natworks) y RNN (Recursive Neural Networks), muy utilizadas en la literatura y por grupos de investigación en el área de reconocimiento de objetos hechos por el ser humano.

7.1 Vector característico

En la Sección 6.3 del Capítulo 6, se analizó la construcción de primitivas tridimensionales a partir de una nube de puntos. Cada primitiva consistía en una serie de parámetros que permitían hacer la construcción tridimensional de la forma que mejor ajustaba a un conjunto de puntos de entrada. Esto permitía describir la forma, la posición, la rotación y la escala del objeto. Son estos parámetros los que se utilizarán como atributos para construir el vector característico.

El uso de los parámetros de las ecuaciones, como características reconocibles de un objeto en un proceso de reconocimiento visual, es uno de los principales aportes de esta tesis. Si bien desde los noventa existen trabajos que prueban la eficiencia del reconocimiento basado en modelos tridimensionales (formados por primitivas tridimensionales) [Arman and Aggarwal, 1993], [Johnson and Hebert, 1999], estos se basan en el emparejamiento de puntos con posibles vistas de un modelo tridimensional previamente construido y guardado en una base de conocimiento. El problema con este modelo en el reconocimiento de objetos es que cada punto escaneado se empareja con posibles posiciones del modelo tridimensional guardado, haciendo de esto un procedimiento sumamente tardado y con resultados que, inclusive en trabajos actuales [Bosché, 2010], no sobrepasan el 85 %.

Para mejorar los problemas de tiempo de cómputo y porcentajes de asertividad, el presente trabajo ocupa los parámetros de las ecuaciones de las formas, que son invariantes a traslaciones, rotaciones y escalas entre las partes de un mismo objeto.

Trabajos, como los presentados en [Li et al., 2011] y [Le and Duan, 2017], demuestran cómo los objetos cotidianos hechos por el ser humano pueden ser representados por una serie de primitivas tridimensionales, que se adaptan al diseño industrial de la mayoría de las piezas en nuestro entorno. Si bien estos trabajos ofrecen una gama amplia de primitivas tridimensionales como esferas, cilindros, conos, prismas rectangulares, toroides, prismas trapezoidales, dodecaedros, octaedros, entre otros, como demuestra [Georgiev et al., 2016], la mayoría de los objetos en una escena común puede ser representada por esferas, prismas rectangulares, cilindros y conos, como en nuestro caso.

El vector característico se compondrá entonces de los parámetros de las ecuaciones de esferas, prismas rectangulares, cilindros y conos, pero queda abierta la pregunta ¿Cuántas primitivas tridimensionales de cada categoría conformarán el vector característico? Lamentablemente, al momento de escribir este trabajo, no existe un documento que especifique el número de primitivas tridimensionales que permite formar objetos cotidianos, por lo que es labor de esta tesis establecer un número de objetos identificables para construir el vector característico. Para esta labor, se toma como referencia el banco de datos que se desglosa en la Sección 7.2, conformado por 50 objetos de uso común hechos por el ser humano. De estos, se han obtenido las primitivas tridimensionales que los conforman al ejecutar el algoritmo de ajuste de primitivas tridimensionales, descrito en el Capítulo 6.

7.1.1 Análisis de cilindros en objetos hechos por el ser humano

El Capítulo 6, dedicado al ajuste de primitivas tridimensionales, toma como salida, la segmentación convexa obtenida en el Capítulo 5, i.e., traduce a las primitivas tridimensionales que mejor se ajustan a un conjunto de puntos convexo. Además, lleva a cabo esta tarea de forma automática, por lo que para determinar el número de cilindros identificables en un objeto hecho por el hombre, sólo se ejecutará el procedimiento propuesto en el Capítulo 6 sobre los objetos descritos en el banco de datos de la Sección 7.2, lo que da como resultado la gráfica de la Figura 7.1.

A partir de la Figura 7.1, se puede notar que existe una gran cantidad de objetos hechos por el ser humano que no poseen cilindros; aquellos que los poseen, lo hacen en general entre 1 y 2 solamente, siendo la minoría aquellos que poseen 3 o 4; de estos últimos, se podría pensar en sillas o rejas, aunque parecen objetos con pocas ocurrencias. Por lo tanto, para compactar el banco de datos, sólo se hara uso de 2 cilindros para el vector característico (aunque este número puede ampliarse en versiones posteriores). Recordando la Ecuación 6.12, un cilindro se compone de los parámetros: centro de la base (x, y, z), vector normal de la base (x, y, z), altura, radio 1 y radio 2; se ha omitido intencionalmente el centro del circulo superior, así como su normal, ya que son datos derivables de los anteriormente mencionados, por lo que

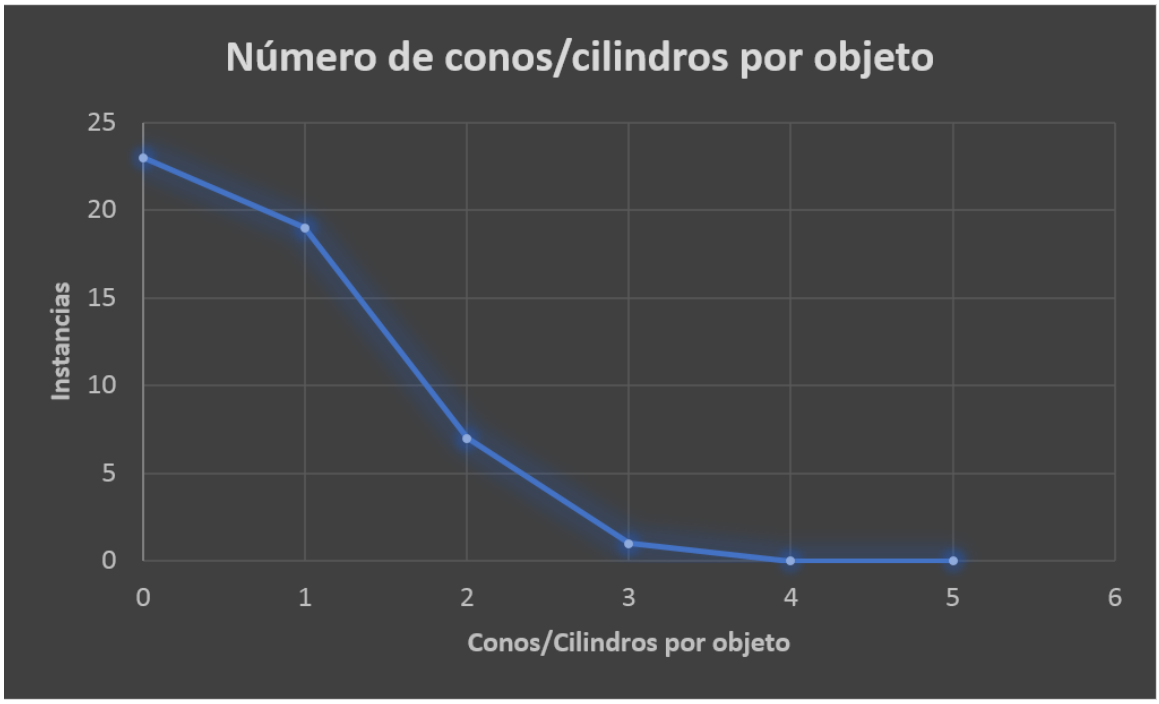


Figura 7.1: Instancias de cilindros encontrados en el banco de datos descrito en la Sección 7.2

Tabla 7.1: Parámetros del cilindro para el vector característico. Primero, las coordenadas del centro de la base $c(x, y, z)$, seguidas del vector normal al círculo de la base $n(x, y, z)$, después la altura h y finalmente el radio de la base r_1 y el radio de la tapa superior r_2.

CILINDRO / CONO								
c_x	c_y	c_z	n_x	n_y	n_z	h	r_1	r_2

no aportarían información relevante al sistema. Cada cilindro queda representado entonces por 9 atributos, como se muestra en la Tabla 7.1.

7.1.2 Análisis de prismas rectangulares en objetos hechos por el ser humano

Siguiendo el mismo análisis presentado en la Subsección 7.1.1, se ejecuta el procedimiento presentado en el Capítulo 6, pero esta vez teniendo en cuenta el número de prismas rectangulares que aparecen durante el ejercicio. El resultado es presentado en la gráfica de la Figura 7.2.

La Figura 7.2 revela que los prismas rectangulares son una de las primitivas más usadas en el diseño industrial. Mayoritariamente, las formas tridimensionales cuentan con un prisma, pero puede haber ocurrencias de figuras con hasta cuatro prismas rectangulares. Por el número de apariciones, se dejará espacio para cuatro prismas

Figura 7.2: Instancias de prismas rectangulares encontrados en el banco de datos descrito en la Sección 7.2

Tabla 7.2: Atributos del prisma rectangular para el vector característico. Primero, las coordenadas del vértice de origen $c(x, y, z)$, enseguida las coordenadas del vector $W(x, y, z)$ del sistema ortonormal y finalmente, las tres dimensiones del prisma w, h, d.

PRISMA RECTANGULAR								
c_x	c_y	c_z	W_x	W_y	W_z	w	h	d

rectangulares en el vector característico. La Ecuación 6.19 presenta varios parámetros para definir un prisma rectangular: tres vectores que forman un sistema ortonormal (W, H, D) y tres valores dimensionales que acotan los vectores (w, h, d), así como una esquina de origen $C(x, y, z)$ ubicada en la parte inferior-derecha-delantera. De la triada de vectores (W, H, D), podemos quedarnos sólo con uno de los vectores, ya que al formar un sistema ortonormal, según las restricciones de la Ecuación 6.19, la información de los otros dos vectores resulta redundante. Se escogerá el vector W sólo por ser visualmente más atractivo, aunque cualquiera de los vectores del sistema ortonormal podría ser usado. Cada prisma rectangular será representado en el vector característico por una esquina de origen $c(x, y, z)$, un vector del sistema ortonormal $W(x, y, z)$ y una triada de dimensionalidad del prisma rectangular (w, h, d), tal como se muestra en la Tabla 7.2.

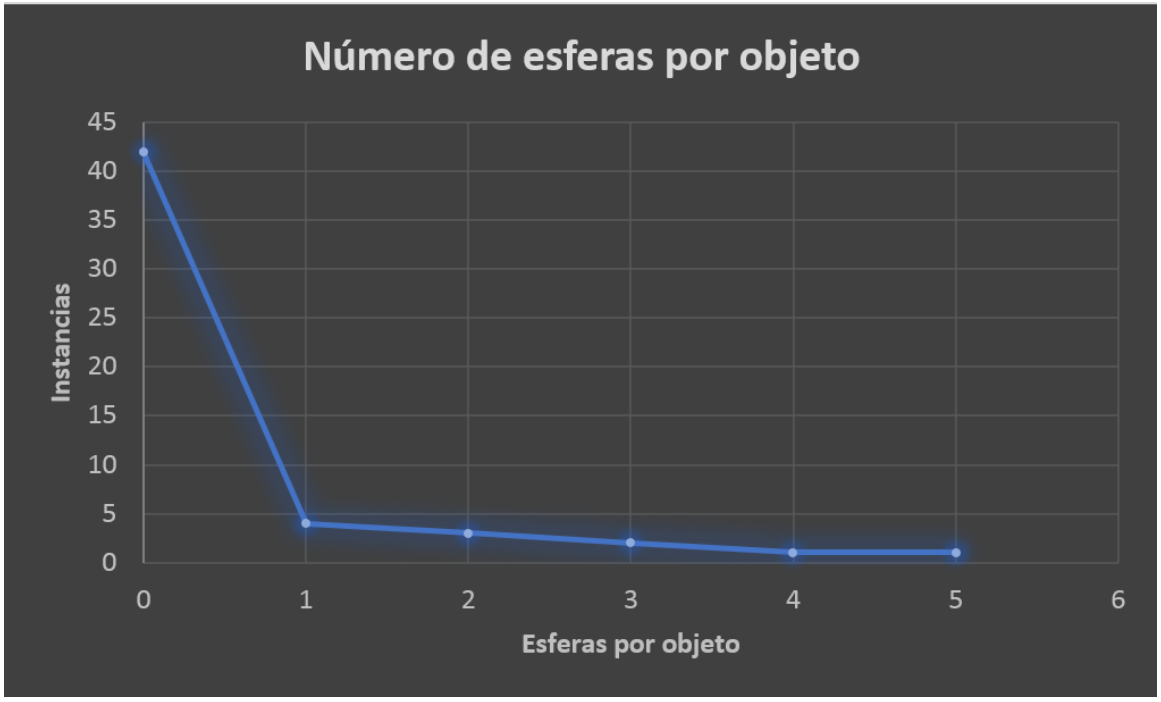


Figura 7.3: Instancias de esferas encontradas en el banco de datos descrito en la Sección 7.2

Tabla 7.3: Atributos de la esfera para el vector característico. En este caso, la representación es posible usando sólo una coordenada espacial para el centro $c(x, y, z)$ y un radio r.

ESFERA			
c_x	c_y	c_z	r

7.1.3 Análisis de esferas en objetos hechos por el ser humano

Una vez más, se aunará el banco de datos presentado en la Sección 7.2 y el procedimiento descrito en el Capítulo 6, buscando esferas dentro de los objetos de interés. Como se aprecia en la Figura 7.3, se ha encontrado una menor cantidad de esta primitiva tridimensional dentro del banco de datos.

El resultado de analizar la Figura 7.3 es que, en general, en el espectro de objetos hechos por el ser humano, que se usó para el experimento, existen pocas esferas. Generalmente, se puede ver sólo una o dos esferas en los objetos analizados. Con el fin de no crear espacios vacíos en el banco de datos, se incluirá un espacio únicamente para dos esferas. A partir de la Ecuación 6.9, se vislumbra que para cada esfera se necesitan cuatro atributos: una coordenada espacial que representa el centro de la esfera $c(x, y, z)$ y un radio r, como se muestra en la Tabla 7.3.

Tabla 7.4: Forma final del vector característico. Los 62 atributos se obtienen al expandir esta tabla con las definiciones de las Tablas 7.1, 7.2 y 7.3.

Vector característico							
Cono 1	Cono 2	Prisma 1	Prisma 2	Prima 3	Prisma 4	Esfera 1	Esfera 2

7.1.4 Forma del vector característico

Del análisis de las Figuras 7.1, 7.2 y 7.3, es claro que la mayoría de los objetos se componen por unas pocas primitivas tridimensionales. Esto cobra sentido al comprarlos con la vida real, ya que es posible percatarse de que los objetos hechos por el ser humano pueden ser reducidos a un conjunto simple de primitivas tridimensionales. En general, en objetos hechos por el ser humano, es posible encontrar varias primitivas tridimensionales dentro de los ornamentos dichos objetos, pero estos son más pequeños que la forma en general. Recordando el Algoritmo 6, los segmentos pequeños o que no tienen suficiente profundidad son ignorados y pasan a ser parte de un segmento convexo más amplio.

El vector característico queda descrito por nueve atributos para cada cilindro, considerándose dos cilindros, nueve atributos para cada prisma rectangular, habiendo cuatro prismas rectangulares y cuatro atributos para cada esfera, con dos esferas por vector, dando un total de 62 atributos, los cuales están descritos en la Tabla 7.4.

7.2 Banco de datos

Para facilitar la investigación en el campo del reconocimiento de objetos, en especifico en objetos hechos por el hombre, industriales, urbanos y de interiores, uno de los objetivos de esta tesis es crear un banco de datos útil a técnicas que busquen, al igual que nosotros, crear el ajuste de primitivas tridimensionales como método para definir el vector característico. Si bien dentro de los dos principales repositorios de bancos de datos para visión por computadora, CVonline: Image Databases[1] y Yet Another Computer Vision Index To Datasets (YACVID)[2], existen al momento de escribir esta tesis, 50 bancos de datos de imágenes RGB-D, 21 de ellos son datos espaciales de actividades humanas, estimación de poses o identificación humana, por lo que no son útiles a la presente investigación.

Del resto de trabajos, se tienen opciones interesantes, así como trabajos similares al nuestro, e.g., trabajos en RGB-D enfocados a la identificación de objetos hechos por el ser humano y con un mínimo de 40 clases. Otro ejemplo es el banco de datos de objetos tridimensionales de ModelNet [Wu et al., 2015] con nubes de puntos de objetos en interiores; también, bajo solicitud, se ha podido acceder al banco de datos de

[1] http://homepages.inf.ed.ac.uk/rbf/CVonline/Imagedbase.htm
[2] http://riemenschneider.hayko.at/vision/dataset/

[Redmon and Angelova, 2015] y en el caso de *sketches*, se dispone de manera pública del banco de datos propuesto por [Eitz et al., 2012].

De los tres bancos de datos citados con anterioridad, se encontró que los tres ya tenían un tratamiento previo, ya fuera por operadores o descriptores, lo cual los desestimaba inmediatamente. Esta situación se repetía para el resto de las opciones disponibles por una o varias de las siguientes causas:

1. El número de clases era inferior a 40.
2. El banco de datos tiene un tratamiento previo mediante un descriptor o algún otro operador.
3. Se incluyen plantas, animales u otros objetos orgánicos.
4. El banco se compone únicamente por algunas pocas escenas 3D con una gran cantidad de objetos (más utilizados para segmentación)
5. No se incluyen múltiples vistas del mismo objeto, sino sólo la misma vista de objetos similares que caen en la misma clase.
6. Se incluyen únicamente datos de profundidad o bien RGB, pero no ambos.

Cualquiera de los puntos antes mencionados provoca que el banco de datos analizado no sea útil para el presente trabajo (siendo el más recurrente el punto 5). De aquí, la necesidad de presentar un nuevo banco de datos que ayude a más investigadores en el área a probar, mejorar y proponer nuevas metodologías bajo el enfoque del primitivas tridimensionales ajustadas, como atributos del vector característico.

7.2.1 Descripción del banco de datos

Es posible usar los bancos de datos de [Redmon and Angelova, 2015] y ModelNet [Wu et al., 2015] como base para construir nuestro banco de datos. Sin bien, el vector característico se forma por 62 atributos (como se mencionó en la Sección 7.1.4), mientras los trabajos antes citados se componen de imágenes o descripciones completas, llegando a cientos o miles de características. Lo que resulta interesante para el presente trabajo es retomar los objetos que abordan los dos bancos de datos mencionados previamente.

La siguiente es una lista de los 50 objetos hechos por el ser humano que se tomaron en consideración para este acercamiento. Fueron seleccionados a partir de los bancos de datos de [Redmon and Angelova, 2015], enfocado únicamente en objetos hechos por el ser humano, y de [Deng et al., 2009], que es un banco de datos mucho mayor que el anterior, pero tiene una miscelánea de objetos que también incluye animales y plantas. Se consideraron únicamente los primeros 50 objetos a los que se tuvo acceso inmediato, pero al ser aleatorio el orden de los bancos de datos, esto no afecta de ninguna manera el resultado de la clasificación:

1. Bote de basura grande

2. Bote de basura pequeño
3. Sillón
4. Ventilador
5. Teléfono
6. Secadora
7. Cámara
8. Laptop
9. Calentador
10. CPU
11. Pantalla 24”
12. Pantalla 15”
13. Dispensador de agua
14. Kleenex
15. Bocina
16. SubWoofer
17. Impresora
18. Balón de Basquetabll
19. Smarthphone
20. Tablet
21. Carpeta
22. Libro pequeño
23. Libro grande
24. Proyector
25. Cubeta
26. Vaso grande
27. Vaso pequeño
28. Cajón
29. Disco duro
30. Doraemon
31. Muñeco
32. Teléfono antigüo
33. Teléfono inalámbrico

34. Audífonos
35. Regulador
36. Lata de refresco
37. Botella de cerveza
38. Linterna de mano
39. Lampara de escritorio
40. Pintura en spray
41. Cargador de laptop
42. Router
43. Maletín
44. Maceta grande
45. Maceta pequeña
46. Cinta adhesiva
47. Microondas
48. Escritorio
49. Taza
50. Cojín

La importancia del método propuesto en este trabajo recae en cómo las primitivas tridimensionales se ajustan al objeto. Si bien no se requieren muchas instancias o fotografías RGB-D del mismo objeto, si se necesitan imágenes tomadas desde diferentes ángulos del objeto que se va a describir. De esta forma, el objeto puede ser identificado aún cuando esté rotado, respecto a su vista principal. Esta es la principal razón de crear un nuevo banco de datos.

7.2.2 Construcción del banco de datos

Es importante mencionar que la figura descrita por el vector característico, primero debe centrarse, i.e., se hace coincidir su centro tridimensional con la coordenada del origen. Esto no conlleva ningún problema, pues dada la descripción de primitivas basta con fijar los límites superiores e inferiores de cada eje coordenado para conocer el centro de la imagen. También se hará una normalización de los ejes coordenados, de forma que cada uno vaya de -1 a 1. Dado que Kinect® tiene datos precisos de sus valores de alcance, es posible fijar los límites de los tres ejes coordenados a conveniencia. Para esta labor, se establecen los límites de todos los ejes en 3 metros, que pasarán a ser coordenadas de -1 a 1 en el proceso de normalización.

El SDK de Kinect® presenta una utilidad para obtener las coordenadas "reales" del espacio, a partir de las coordenadas del sistema manejado por Kinect®. Esta

transformación es sólo una conversión e interpolación de valores, como se explica en [Kowsari and Alassaf, 2016]. Mediante la utilidad de conversión, se traducen a milímetros las coordenadas de los parámetros de las ecuaciones que describen las primitivas tridimensionales del objeto. Este proceso es muy útil, ya que los valores de las ordenadas y abscisas cambiarán a diferentes profundidades; gracias a esta medición, es posible trabajar sobre la medida del objeto que, aunado al proceso de centrado, hace que el banco de datos sea invariante a posiciones.

Como se aprecia en la Figura 7.4, para la construcción del banco de datos se tomarán 20 muestras de cada uno de los objetos que se desea clasificar. En este caso, se ilustra una silla ejecutiva que se forma por varios prismas, cilindros y esferas, siendo uno de los objetos más completos en el banco de datos, respecto al número de primitivas que lo componen. Si bien se podría pensar que estas pueden ser pocas muestras, la diferencia con otros trabajos recae en que, para este tipo de banco de datos, la asertividad no variará demasiado a medida que más patrones por objeto se introduzcan. La razón de esto es que cada nuevo ángulo con el que se hace una medición, aporta nueva información. Sin embargo, cuando el ángulo de medición es muy similar a uno de los ya existentes, la información del sistema no varía, ya que se obtendrán prácticamente las mismas primitivas tridimensionales. Durante los experimentos de medición con Kinect®, se determinó que en variaciones menores a 45°, no se presentaban cambios importantes en las primitivas tridimensionales que forman un objeto. Usando la misma lógica, se varía el ángulo en la vertical desde donde se mira el objeto (de frente, 45° desde arriba y 45° desde abajo). No se varía más, ya que se tendrían vistas aéreas y completamente desde abajo, que harían ver a todos los objetos como cajas o cilindros.

Con la lógica del párrafo anterior, podríamos pensar que el número idóneo de muestras es de 24. Sin embargo, experimentalmente este número es menor pues, aunque la condición para que varíen las primitivas tridimensionales es un ángulo de 45°, esta es una cota inferior. Experimentalmente, basta con 20 muestras para representar la variación de las primitivas tridimensionales, pero este es un trabajo que debe ser realizado por el ser humano para eliminar las muestras repetidas o muy similares.

Un punto importante a tener en consideración es que, en varias ocasiones, el algoritmo encontrará más primitivas tridimensionales de las que pueden ser almacenadas en el banco de datos. De la Sección 7.1.4, es importante recordar que sólo se ha dejado espacio para dos cilindros/conos, cuatro prismas rectangulares y dos esferas, pudiéndose dar los casos en que pequeños segmentos convexos son detectados y ajustados como una primitiva tridimensional. Esto ocurre por errores en el procesamiento o en las mediciones, o bien porque tales pequeños segmentos convexos son parte del ornamento del objeto, pero es necesario recordar que lo que se busca con la presente propuesta es describir al objeto en su estructura más general. La idea de expresar un objeto como la generalidad de sus primitivas tridimensionales se traduce en aprovechar únicamente las primitivas tridimensionales de mayor tamaño, que describen mayoritariamente al objeto. Por esta razón, sólo se conservarán las n primitivas más relevantes en cuanto a tamaño, donde n es el número de primitivas aceptadas en el vector característico.

Figura 7.4: Muestras que forman la descripción del objeto "sillón" en el banco de datos. De izquierda a derecha se encuentra la imagen a color, la imagen de profundidad y finalmente la imagen procesada, donde en verde se representan las esquinas del grafo de convexidad.

Método de validación	Número de apariciones
2-Fold Cross Validation	3
5-Fold Cross Validation	12
10-Fold Cross Validation	**15**
Leave One Out	7
Hold Out	3
Curvas ROC	15
Boot Strap	2
WAIC	1

Tabla 7.5: Número de apariciones de los métodos de validación desde 2016 a 2018 en artículos científicos de clasificación de objetos.

Las primitivas tridimensionales, que pasen a ser parte del vector característico, se introducen en orden de volumen, i.e., primero las que posean un mayor volumen y al final aquellas de menor volumen; esta condición es simple pero importante, ya que sin ella la asertividad podría decrecer considerablemente porque, al ver un objeto en distintas posiciones, es probable que se detecte una primitiva tridimensional antes que otra. Si no tienen un orden, se provocarían grandes variaciones en cada atributo, pero gracias al orden, se mantiene una generalidad del objeto.

Con 20 mediciones por cada objeto mencionado en la Sección 7.2.1, se tiene un total de 1,000 mediciones de objetos, cada una de las cuales se compone de 62 atributos. Todas ellas se han procesado tal como se muestra en la Figura 7.4. Con estas mediciones, el banco de datos se encuentra listo para comenzar los experimentos de clasificación.

7.3 Resultados de la clasificación

Medir la asertividad de nuestra propuesta conlleva un estudio del estado del arte, de manera que nuestros resultados puedan ser comparados con la mayor cantidad investigaciones en el área utilizando la misma metodología y con los mismos parámetros, por ello durante esta sección además de mostrar los resultados de la asertividad de nuestra propuesta, mostraremos la comparativa con otros trabajos de investigación mediante una metodología de validación seleccionada para este propósito.

7.3.1 Estudio de la metodología de validación

Para determinar el porcentaje de asertividad de la presente propuesta, se hará uso del enfoque *K-Fold Cross Validation* con $K = 10$, siendo el mayormente utilizado en 58 artículos relacionados con el reconocimiento de objetos tridimensionales desde 2016 hasta 2018, como se muestra en la Tabla 7.5.

Método de clasificación	Porcentaje de asertividad
1NN	99.18 %
3NN	94.34 %
5NN	90.94 %
Nayve Bayes	93.15 %
Bayes Network	95.35 %
C45 Tree	99.19 %
Random Forest	99.80 %
Multi-Layer Perceptron	97.40 %
RBF Network	99.59 %
SVM Lineal	98.32 %
SVM Polynomial	99.22 %
SVM RBF	**99.81 %**
SVM Sigmoide	82.14 %

Tabla 7.6: Comparación de la asertividad de diferentes métodos de clasificación con el banco de datos generado en la Sección 7.2.2.

En esta investigación, no se tomaron en cuenta trabajos relacionados con la detección de objetos en imágenes médicas, de cultivos o aeroespaciales, ya que es bien sabido que las curvas ROC (Receiver Operating Characteristic) dominan estos campos, pero no aportan ningún punto comparativo de interés con nuestro trabajo. Se tomaron en cuenta únicamente trabajos de clasificación de objetos urbanos, hechos por el ser humano, de navegación automática en interiores y exteriores, que son los campos donde encontramos puntos de comparación y donde aportamos nuevos resultados. El método de K-Fold Cross Validation fue el más utilizado con 30 apariciones en los artículos estudiados. Además, si bien con $K = 5$ y $K = 10$ se obtuvieron resultados muy similares, $K = 10$ fue el más utilizado.

7.3.2 Estimación de resultados

Dado que el fin de esta tesis no es proponer un método de clasificación novedoso, sino que uno de los objetivos es utilizar la información de reconstrucción para formar un vector característico, se hará uso del programa Weka versión 3.8.2, con el cual se puede probar el banco de datos formado en la Sección 7.2.2 con diferentes métodos de clasificación. Haciendo uso de nuestro estudio comparativo entre métodos de validación presentado en la sección 7.3.1, se hará uso de *K-Fold Cross Validation* con $K = 10$ para todos los métodos que se validarán con Weka. Los resultados de esta ejecución se muestran en la Tabla 7.6.

Como se observa en la Tabla 7.6, los mejores resultados son obtenidos por las máquinas de soporte vectorial (SVM por sus siglas en inglés) con un *kernel* RBF. Sin embargo, al cambiar el *kernel* por uno tipo sigmoide, se obtienen los peores resultados de la Tabla

con un 82.4 % , cifra que, por si sola, resulta competitiva. Utilizando únicamente los resultados de las máquinas de soporte vectorial con *kernel* tipo RBF, se ha calculado la matriz de confusión con ayuda de Weka. Siendo una cantidad considerable de clases, se ha optado por una representación gráfica de la información que permite apreciar rápidamente los cambios en la matriz de confusión, la cual se muestra en la Figura 7.5.

Al analizar la matriz de confusión de la Figura 7.5, es notorio que los principales errores ocurridos se dan entre objetos hechos por el ser humano, que se conforman por un solo prisma rectangular. Se observan casos como la confusión entre un microondas y un CPU o entre una pantalla pequeña y un *subwoofer*. Lo mismo pasa entre una secadora y un bote de basura pequeño, ya que en cierto ángulo existen primitivas que no serán visibles y que se pueden confundir con otros objetos.

7.3.3 Estudio comparativo

Después de una extensa investigación sobre el estado del arte de la clasificación de objetos hechos por el ser humano, existen varios trabajos contemporáneos con los que esta tesis se puede comparar. Los más relevantes debido al estudio que hacen sobre objetos hechos por el ser humano, así como la clasificación que realizan haciendo uso de la extracción de características, a partir de una nube de puntos tridimensionales, son los siguientes:

- El trabajo de [Redmon and Angelova, 2015] realiza la clasificación de objetos hechos por el ser humano, utilizando *Convolutional Neural Networks* y *Grasps*. Este trabajo alcanza una asertividad del 85 % para 240 objetos y usa *5-Fold Cross Validation* para la comparación. El enfoque de redes neuronales convolusionales permite incluir el procesamiento de una imagen (descrito en el Capítulo 4) en capas intermedias de una red neuronal. Dichas capas actúan como filtros o máscaras de la imagen, por lo que los procesos descritos en las Secciones 4.1, 4.2 y 4.3 podrían incluirse en un clasificador basado en redes neuronales convolusionales. Sin embargo, los procesos descritos en la Sección 4.4 y en todo el Capítulo 5 carecen de una representación directa de máscara para el procesamiento de la imagen, razón por la cual no pueden ser definidos por una red neuronal convolusional. Así, este enfoque queda limitado en comparación con nuestra propuesta.
- El trabajo de [Serna and Marcotegui, 2014] hace la detección de objetos hechos por el hombre en ambientes urbanos, enfocándose en objetos que pueden ser encontrados en las calles y que son útiles en la navegación automática. Para ello utiliza morfología matemática. El resultado probado con *10-Fold Cross Validation* y una máquina de soporte vectorial, como clasificador, es de 82 % para 20 clases.
- El trabajo de [Qi et al., 2017] alcanza asertividad del 89 % para 40 clases al identificar objetos hechos por el ser humano de entre una nube de puntos tridimensionales. Este resultado es alcanzado, usando *10-Fold Cross Validation*.

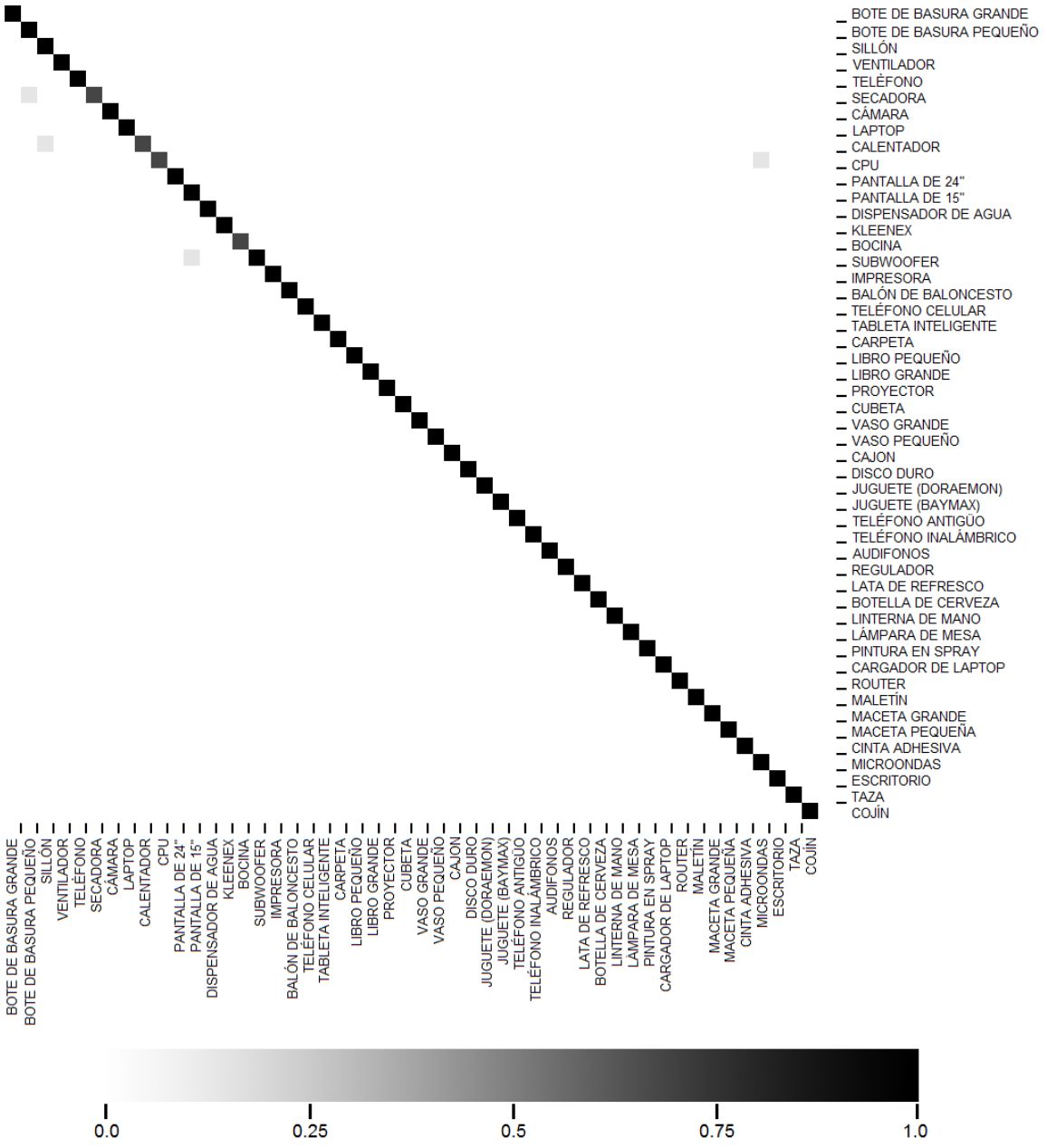


Figura 7.5: Matriz de confusión del banco de datos de la Sección 7.2.2 al aplicar, como método de clasificación, una máquina de soporte vectorial con *kernel* RBF.

> Este trabajo también realiza una segmentación de objetos, de una manera muy similar a la nuestra, y utiliza *Deep Learning* con *Recursive Neural Networks* así como descriptores globales.

Otro trabajo que resulta interesante analizar es el propuesto por [Pang et al., 2015], que hace el ajuste de primitivas tridimensionales (las mismas que se usan en este trabajo) sobre objetos hechos por el ser humano para posteriormente clasificarlos, utilizando una máquina de soporte vectorial. Para crear el vector característico, este trabajo utiliza un descriptor conocido como *Fast Point Feature Histogram* (FPFH) [Rusu et al., 2009], reportando un índice de asertividad del 85.5 %. Este trabajo no muestra el número de clases utilizadas o la técnica de validación, lo que resulta en un inconveniente, dado su parecido con nuestra propuesta.

Las comparaciones anteriores se hacen respecto al método de validación y a la acertividad obtenida. Hablar de tiempos y complejidad computacional resulta poco viable, pues según la metodología ocupada, la complejidad computacional se puede disparar. De hecho, en materia de clasificación se prefiere una mejor acertividad frente a un costo computacional alto (a menos que se busquen aplicaciones de respuesta en tiempo real). En el caso del trabajo de [Redmon and Angelova, 2015], el uso de redes neuronales computacionales es un conocido cuello de botella pues, como se presenta en el trabajo de [He and Sun, 2015], la complejidad es de $O(\sum_{l=1}^{d} n_{l-1} \cdot s_l^2 \cdot n_l \cdot m_l^2) \approx O(n^6)$ donde l es el índice de la capa convolusional, d es la profundidad (el número de capas convolusionales), n_l es el número de entradas de la capa y m_l es el tamaño espacial de la salida. Este tiempo podría parecer excesivo, pero [Redmon and Angelova, 2015] hace uso de una tarjeta Nvidia K20 con 2496 núcleos y una reducción de las imágenes a un tamaño 224 x 224 píxeles para hacer un reconocimiento en sólo 76 milisegundos, mientras que una metodología muy similar, como la que propone [Lenz et al., 2015], tiene un tiempo de procesamiento de 13.5 segundos al correr en un CPU sin reducción en el tamaño de las imágenes.

En el caso del enfoque presentado por [Serna and Marcotegui, 2014], los autores reportan tener un tiempo de ejecución promedio de 1 minuto al ejecutarse en una computadora con 4 núcleos con una velocidad de procesamiento de 2.4 GHz para imágenes de 150,000 puntos y de 390 x 390 píxeles aproximadamente. Si bien no se menciona la complejidad computacional de la propuesta, es posible inferir del trabajo de [Dokládal and Dokladalova, 2011] que se tiene una complejidad aproximada de $O(n)$.

En el trabajo de [Qi et al., 2017] se maneja un enfoque de *Deep Learning* que tiene una complejidad computacional reportada de $O(n^3)$ con la capacidad de procesar 1,000,000 de puntos por segundo. Esto hace posible que, en imágenes de una resolución aceptable como 1000 x 1000 píxeles, se pueda procesar una imagen por segundo, pero en imágenes de 250 x 250 píxeles se pueden procesar hasta 16 imágenes por segundo, utilizando un GPU de 1080 núcleos.

Del análisis expuesto en los tres párrafos anteriores, se puede ver la dificultad de comparar los trabajos en cuestión de tiempos de ejecución, pues algunos usan GPUs

Trabajo	Número de Clases	Asertividad
Redmon et al. 2015	240	85 %
Serna et al. 2014	20	82 %
Qi et al. 2017	40	89 %
Nuestra propuesta (SVM Kernel RBF)	**50**	**99.81 %**
Nuestra propuesta (SVM Kernel Sigmoide)	50	82.14 %

Tabla 7.7: Comparación del porcentaje de asertividad entre trabajos recientes en el reconocimiento de objetos hechos por el ser humano, en específico aquellos con enfoques similares al nuestro.

y otros usan procesadores convencionales. Esto influye en gran medida en que se tengan tiempos bajos, aún cuando la complejidad computacional sea alta. En nuestro caso la complejidad computacional es de $O(nLog(n))$ con un tiempo de procesamiento de 1.7 segundos por imagen en un procesador de cuatro núcleos a 3.0 GHz.

Como se puede ver en la Sección 7.3.1, es posible hacer la consulta de 58 estudios de reconocimiento de objetos hechos por el ser humano, de indumentaria urbana y de interiores, pero consideramos que los trabajos citados en esta subsección son los más representativos, tomando en cuenta la similitud y la asertividad. Un resumen de los resultados se puede encontrar en la Tabla 7.7. Es notorio que con 50 clases, nuestros resultados son superiores a los de otros algoritmos, pero se debe tener en cuenta que es necesario ampliar el número de clases hasta alcanzar resultados asintóticos a una cota.

Capítulo 8

Conclusiones y trabajo a futuro

Como se mencionó en la Subsección 1.3.1, el objetivo general de este trabajo es desarrollar un sistema que abarque los procesos de reconstrucción y clasificación de objetos fabricados por el ser humano, integrándolos en un proceso que aproveche las características de ambos enfoques, siendo robusto al problema de la invariancia, gracias al uso de sensores RGB-D que procesan las imágenes como *sketches*.

El cumplimiento de este objetivo no ha sido trivial. El mero uso de imágenes RGB-D no garantizaba la resolución del problema de la invariancia, el ajuste de primitivas o el incremento de la asertividad en la clasificación. Por ello, una serie de algoritmos presentados en este trabajo fueron programados en Matlab 2016b para mantener la homogeneidad y ayudar en la sencillez de la programación. Algunos métodos para mejorar la calidad de las imágenes del sensor fueron presentados en las Secciones 3.1 y 3.2. Mediante ellos, se eliminaban los huecos de las imágenes de profundidad, debidos a las luces y sombras infrarrojas naturales del ambiente, logrando imágenes más nítidas.

Al pretender tratar con dos espacios distintos (RGB y profundidad) ha sido necesario describir una forma de trasladar los elementos entre un espacio y otro; esta transformación fue descrita en la Sección 3.3. Mediante una forma sencilla de pasar las mediciones de cada imagen de un espacio a otro, podíamos resolver el problema de la invariancia, de una manera novedosa: el uso de *sketches*. La gran diferencia entre obtener únicamente los contornos de un objeto y automatizar un proceso de *sketching* radica en la generalización de la forma de dicho objeto; en el Capítulo 3, la diferencia resulta más notoria. El proceso de extracción del fondo de las imágenes descrito en las Secciones 4.1 y 4.2 es más sencillo que en los trabajos relacionados. De igual manera, en la Sección 4.3 se presentó un algoritmo que permite manejar la extracción de contornos y esquinas, de una manera más rápida que la usual; ambas técnicas, la de extracción de fondos y la de búsqueda de contornos y esquinas, se llevan a cabo en sus respectivos espacios, pero es el emparejamiento de resultados, expuesto en la Sección 4.4, el que permite crear los *sketches*, pues unir ambos espacios permite la invariancia ante iluminaciones, sombras, texturas y colores. Este proceso de emparejamiento únicamente da como resultado una única serie de contornos y esquinas que

pertenecen al *sketch*.

A partir de este momento, el presente trabajo se desprende un poco de la representación gráfica de los resultados para pasar a una representación matemática, primeramente al convertir la serie de contornos y esquinas en un grafo, como se describió en la Sección 5.1. Este grafo conecta esquinas entre sí, cuando se puede llegar a ellas mediante un camino formado por los contornos del objeto. Esta entidad matemática es una representación del objeto que se quiere caracterizar mediante primitivas tridimensionales, la cual resulta mucho más sencilla de manejar que las imágenes puras.

Mediante el Algoritmo 5 presentado en la Sección 5.2, es posible separar el objeto en segmentos convexos como se explicó en la Sección 5.3, dando como resultado una serie de conjuntos convexos. La justificación de esta descomposición convexa reside en que todas las primitivas tridimensionales que se manejaron en este trabajo son convexas, por lo que la reconstrucción de los objetos hechos por el ser humano debe ser realizada en segmentos convexos, donde se ajustarán primitivas convexas.

Esferas, conos, cilindros y prismas rectangulares son las primitivas tridimensionales seleccionadas para ajustar los segmentos convexos. En este momento es cuando se retomaron trabajos basados en la reconstrucción de *sketches*, como los de [Chen et al., 2013] y [Shtof et al., 2013]. Estos trabajos usan el Lanragiano aumentado (ver Sección 6.2) para hacer el ajuste de las primitivas tridimensionales y de las funciones objetivo para cada una de las primitivas tridimensionales, descritas en las Secciones 6.3.1, 6.3.2 y 6.3.3. Esto indicaba únicamente como se ajusta una u otra primitiva tridimensional a cada conjunto convexo, es por ello que utilizando el Algoritmo RANSAC (ver Sección 6.1) se generó un proceso que escoge automáticamente la primitiva que mejor ajusta a un conjunto convexo; se trata del Algoritmo 11, presentado en la Sección 6.4. Esto se logró minimizando el error entre el modelo de la primitiva tridimensional ajustada y los puntos del conjunto convexo, mediante una medida para cada primitiva tridimensional. El resultado de esta fase es la primitiva tridimensional que mejor ajusta a un conjunto convexo. Iterando sobre todos los conjuntos convexos se obtienen todas las primitivas tridimensionales que forman un objeto hecho por el ser humano.

En los objetos hechos por el ser humano, las piezas no son independientes, sino que manejan una estructura dirigida por el diseño industrial. Para emular este proceso, las primitivas tridimensionales fueron sometidas a un conjunto de restricciones geosemánticas (ver Sección 6.5). Estas garantizan que existan relaciones de paralelismo, perpendicularidad, colinealidad, concentricidad y coplanaridad entre las piezas, ayudando a que se cierren huecos entre ellas y a que sean dependientes unas de otras.

Matemáticamente, se obtuvo una serie de parámetros de las ecuaciones que describen cada uno de los segmentos convexos que forman los objetos hechos por el ser humano. Estos parámetros constituyen un vector característico, como se describió en la Sección 7.1. Se escogió una cantidad de primitivas tridimensionales, al estudiar con cuidado una serie de objetos hechos por el hombre, formando un vector de dos

conos/cilindros, cuatro prismas rectangulares y dos esferas. Considerando 50 objetos hechos por el ser humano, se creó un banco de datos, con 20 instancias por clase, que corresponden a las diferentes vistas de un objeto. Este proceso de muestreo, a través de diferentes ángulos, más el centrado del objeto, garantizan la invariancia a posición, traslación y rotación.

El banco de datos creado se usó para medir la asertividad de la metodología propuesta al clasificar objetos hechos por el ser humano. Los resultados, como se muestra en la Tabla 7.7, son sumamente competitivos, mejorando la asertividad entre el 10 % y el 15 %, al comparar el presente trabajo con los del estado del arte. Aunado a la mejora en asertividad, se propuso una metodología que es resistente a la invariancia de posición, transalación, rotación, y a problemas de sombras, iluminación, colores y texturas.

8.1 Análisis de la complejidad

De forma premeditada, en las Secciones 4.1, 4.2, 4.3, 4.4, 5.1, 5.2, 5.3 y 6.4, se justificó (o se citó la fuente de justificación) la complejidad matemática del procedimiento descrito, lo que ayuda a formar un fundamento matemático completo de la complejidad computacional de nuestra propuesta. Cabe recordar que, a partir del procedimiento descrito en la Sección 4.4, la entrada deja de ser el número de píxeles o puntos en la imagen y pasa a ser el número de vértices o esquinas, por lo que resulta necesario homologar la nomenclatura para un claro entendimiento de la complejidad. Así, se propone la nomenclatura de n para el número de píxeles de entrada y h para el número de vértices que se procesan, siendo $h << n$. Bajo esta nueva nomenclatura, a continuación se listan las complejidades de todos los procedimientos de nuestra propuesta:

- Extracción del fondo: $O(n)$
- Normalización del histograma de profundidad: $O(L^2)$ siendo L el número de niveles de gris y $L << n$.
- Extracción de contornos y esquinas: $O(n \cdot hLog(h))$
- Emparejamiento de contornos y esquinas. $O(k^2h)$ siendo k el tamaño de la máscara, de modo que $k^2 < h$.
- Grafo de puntos de interés: $O(hLogh)$.
- Descomposición por aproximación convexa: $O(hLogh)$.
- Selección de primitivas tridimensionales: $O(F \cdot Q \cdot k \cdot h) \approx O(h^2)$, siendo F el número de primitivas tridimensionales que forman el objeto, Q el número de primitivas tridimensionales posibles y k el número de iteraciones permitidas.

En la notación *big O*, se considera el término dominante como el que expresa la complejidad de un algoritmo. Así, de la lista anterior, se debe determinar el término

dominante, por lo que es necesario declarar la relación entre h y n. En general, se sabe que $h << n$ y de la Sección 6.4 se conoce que h se encuentra en el segundo orden de magnitud, mientras que en la Sección 4.2 se establece que n se encuentra en el quinto orden de magnitud, por lo que se espera que $h^2 \leq n$. Con ello, la extracción del fondo, así como la de contornos y esquinas, dominan la complejidad de los demás algoritmos, siendo $O(n \cdot hLog(h))$ la complejidad dominante de la propuesta.

Para mantener esta complejidad, es importante la elección correcta de un clasificador. De la Tabla 7.6, se deduce que los mejores resultados se han obtenido con los clasificadores SVM con Kernel RBF y Random Forest. El primero presenta una complejidad $O(n_{sv} \cdot d)$, de acuerdo a [Claesen et al., 2014], donde d representa la dimensionalidad del vector y n_{sv} es el número de vectores de soporte, que también puede interpretarse como el número de clases menos uno. En el caso de los Random Forest, la literatura [Louppe, 2014] menciona una complejidad promedio de $O(MLogN)$, así como la complejidad en el peor de los casos dada por $O(M \cdot N)$, donde N es número de árboles y N es el número de muestras en el banco de datos.

Dados los datos reportados en la Sección 7.2.1, se sabe que $n_{sv} \cdot d < n$ y $M \cdot N < n$ (escogiendo un valor de M adecuadamente menor), lo que ayuda a conservar la complejidad de nuestra propuesta. Sin embargo, al incrementar el número de clases, las SVM con Kernel RBF tienden a mantener su complejidad de forma que $n_{sv} \cdot d < n$, aún en el peor de los casos, mientras que para los Random Forest, se puede lograr con facilidad que $M \cdot N > n$, aunque el caso promedio se mantenga como $MLogN < n$.

8.2 Discusión de resultados

Este trabajo cumplió a cabalidad con los objetivos específicos establecidos en la Sección 1.3.1. De manera puntual, se puede hablar de los siguientes resultados que se emparejan con sus pares en la lista de la Sección 1.3.1.

1. La principal aportación de este trabajo al estado del arte es la automatización de la extracción de primitivas tridimensionales y características que componen un objeto. De esta manera, se logró garantizar la autonomía del sistema, a diferencia de los trabajos de extracción de *sketches* [Shtof et al., 2013] y [Chen et al., 2013] en los cuales nos basamos.

2. Se logró desarrollar un método de extracción automática de *sketches* con base en el Algoritmo [Rosten et al., 2010], el cual permite obtener, de forma sencilla y rápida, esquinas en una imagen. De la misma manera, se adaptó este algoritmo para extraer contornos, de forma rápida, en comparación con las técnicas clásicas. Este procedimiento se aplicó tanto en imágenes RGB como de profundidad, de manera que al buscar emparejamientos entre puntos y contornos de ambas imágenes se obtiene una imagen limpia de la forma general del objeto, i.e., un *sketch* automático.

3. A lo largo del Capítulo 6, se desarrolló una metodología para ajustar primitivas tridimensionales al espacio de un *sketch*, utilizando el método del Langragiano aumentado junto con las funciones objetivo propuestas por [Shtof et al., 2013]. Cabe aclarar que, sin importar la entrada, el método del Langragiano aumentado ajusta la primitiva tridimensional fijada en la función objetivo programada, lo que da como resultado el mejor ajuste a la entrada, i.e., no se hace selección de primitivas, sino que la función objetivo de cada primitiva tridimensional dará por resultado el mejor ajuste de la entrada a la primitiva tridimensional programada.

4. La descomposición de un *sketch* complejo para lograr un ajuste de primitivas tridimensionales en espacios simples y convexos del *sketch*, podía lograrse con técnicas ya conocidas en el estado del arte, como las propuestas por [Liu et al., 2010], [Lien and Amato, 2006] y [Ren et al., 2013]. Sin embargo, el aprovechamiento de las estructuras creadas para el *sketch* facilitaba el cómputo y mejoraba los tiempos de ejecución, por lo que se optó por crear una solución propia descrita en el Algoritmo 6. Una mejora notable es que a partir de este algoritmo se desarrolló una versión que permite hacer la segmentación convexa de objetos con huecos (un tema actual en el estado del arte) lo que se reflejó en el Algoritmo 7.

5. Retomando el segundo punto de esta lista, las funciones objetivo ajustan sus respectivas primitivas tridimensionales a un segmento convexo del *sketch*, pero cada función objetivo es independiente. Por lo tanto, se obtienen varias primitivas tridimensionales con un error asociado, el cual se calcula como la distancia entre el modelo obtenido de la primitiva tridimensional y los puntos de entrada. Este error asociado es el que sirve para seleccionar una primitiva tridimensional de manera automática. Para ello, se utilizó como base el algoritmo RANSAC, iterando sobre porciones de la entrada, i.e., en cada iteración se toma un segmento de puntos y se busca la mejor aproximación. Esto permitió crear un algoritmo tolerante a errores del sensor (un característica sumamente deseable en el caso de Kinect). En cada iteración se obtienen modelos de primitivas tridimensionales y sus errores asociados, y al final se escoge la primitiva tridimensional con el menor error. Este es el procedimiento seguido por el Algoritmo 11.

6. Para garantizar la estructura de la forma que se está construyendo, se necesita una manera de garantizar las restricciones del diseño industrial de los objetos hechos por el ser humano. Para ello, se implementó, como parte de las funciones objetivo del Capítulo 6, las restricciones geosemánticas establecidas por *sketches* [Chen et al., 2013] [Shtof et al., 2013].

7. Dar cumplimiento al objetivo de crear un banco de datos de 50 objetos hechos por el ser humano, fue una labor ardua, aún habiendo programado los algoritmos que permiten crear los vectores característicos, de manera automática. Nuestro banco de datos se compone de 1,000 instancias con 62 columnas cada una, i.e., se recolectaron 62,000 valores de caracterización de objetos formando el

banco de datos descrito en la Sección 7.2.1. Para que sea útil a otros grupos de investigación y para que nuestros resultados puedan ser comprobados, el banco de datos se encuentra en formato CSV, estándar que permite su difusión y tratamiento mediante diferentes programas de clasificación de patrones.

8. Una comparativa entre los diferentes métodos de clasificación se muestra en la Tabla 7.6 que compara varios métodos disponibles en el software Weka en su versión 3.8.2; se tomaron enfoques probabilísticos, redes neuronales, redes Bayesianas, máquinas de soporte vectorial, árboles de decisión (incluyendo Random Forest) y algoritmos perezosos. De entre los enfoques mencionados sobresalieron las máquinas de soporte vectorial, tanto por lograr los mejores resultados en la clasificación, 99.81 % aplicando un kernel RBF, como por tener también los peores resultados, 82.14 % al aplicar un kernel Sigmoide.

9. Entre muchos de los enfoques no existe una significancia estadística, ya que varios se encuentran sobre el 97 %, por lo que aún no se puede hablar de un enfoque claramente superior. Sería necesario incrementar el tamaño del banco de datos para poder determinar un método adecuado. En el alcance de este trabajo, los Random Forest y las máquinas de soporte vectorial con un Kernel RBF se muestran superiores al resto de los enfoques, con 98 % y 98.1 % de asertividad en la clasificación respectivamente, pero sin separarse más de un 2.8 % de la media de los otros enfoques.

10. Para poder hacer la medición de los resultados de clasificación, se hizo una investigación extensa sobre los métodos de medición, analizando los trabajos de los últimos dos años en materia de reconocimiento de objetos urbanos, objetos en interiores y en general objetos hechos por el ser humano. Se tomaron en cuenta un total de 58 artículos, siendo *K-Fold Cross Validation* el método más recurrido por los autores; además el parámetro K se estableció en 10, en la mayoría de los casos. Para poder realizar la comparación de nuestros resultados con el resto de la comunidad, se escogió *K-Fold Cross Validation* con $K = 10$ como método de medición de resultados.

Es posible decir que los tres grandes aspectos que sostienen este trabajo son: 1) extracción automática de *sketches*, 2) el ajuste de primitivas tridimensionales y 3) la clasificación de objetos hechos por el ser humano. La extracción automática de *sketches* es tan amplia que fueron necesarios tres capítulos de este trabajo para explicar adecuadamente la técnica que puede ser generalizada para la automatización de *sketches* (Capítulos de 3 a 5). La extracción de *sketches* se basa en la extracción de contornos y esquinas de los espacios RGB y de profundidad, su comparación y la aplicación de operaciones morfológicas de cerradura para evitar los huecos entre líneas, tan característicos de la extracción de contornos. La separación convexa no resulta visible, ya que crea un grafo de segmentos convexos, pero forma un paso intermedio en la preparación del *sketch* antes de ajustar las primitivas tridimensionales. El ajuste de primitivas retoma el grafo de segmentos convexos y modela primitivas tridimensionales para que ajusten lo mejor posible a un espacio convexo. Esto se ejemplifica en

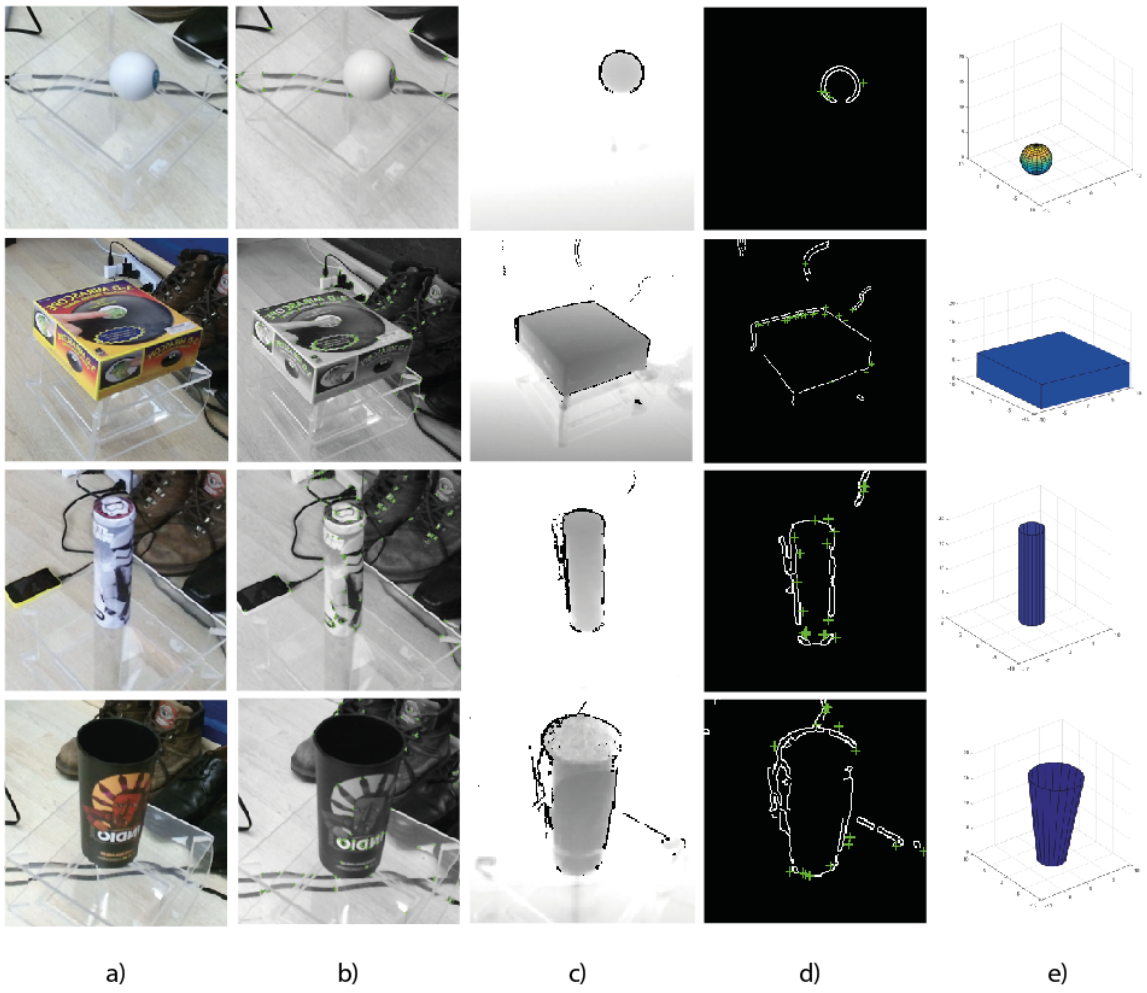


Figura 8.1: Se muestra en a) la imagen original a color y el tratamiento de sus esquinas en b). La imagen de profundidad es mostrada en c), sus esquinas y contornos en d), y finalmente el ajuste de las primitivas correspondientes es mostrado en e).

la Figura 8.1, en la que se han colocado algunos objetos convexos que corresponden a las primitivas tridimensionales. También se puede observar cómo el Algoritmo 11 hace la reconstrucción de la forma.

El mero ajuste de las primitivas tridimensionales a los espacios convexos no es suficiente. De hecho, como se muestra en la Figura 8.2, el ajuste de piezas independientes tiene resultados inesperados. Es por ello que la inclusión de restricciones geosemánticas para objetos industriales es fundamental para esta tesis, logrando formas que siguen los principios del diseño industrial de los objetos con los que se interactúa diariamente.

La Tabla 7.7 refleja la verdadera naturaleza de las restricciones geosemánticas, pues la mejora en los resultados comparados con trabajos relacionados, que hacen uso de ajuste de primitivas o reconstrucciones tridimensionales, se debe a que las primitivas

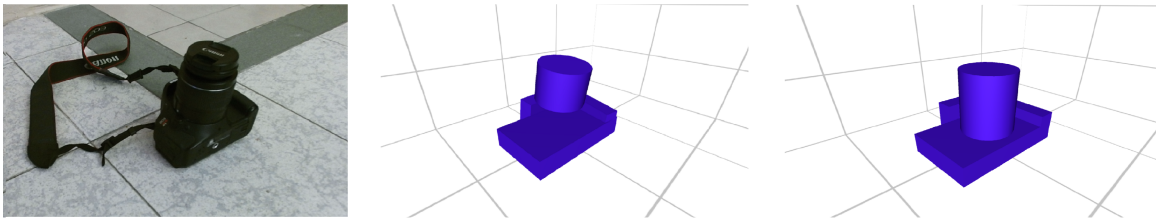


Figura 8.2: En a) se muestra la imagen original en RGB, mientras que en b) se muestra el ajuste de primitivas tridimensionales al cual no se han aplicado las restricciones geosemánticas y en c) se observa la misma reconstrucción al aplicar las restricciones geosemánticas.

tridimensionales son dependientes entre sí gracias a las restricciones geosemánticas, lo que desemboca en modelos estables más fáciles de reconocer.

La clasificación de objetos hechos por el ser humano es el punto final del trabajo. En nuestro caso, los objetos son mejor clasificados gracias a los modelos estables y a las características de forma, no a los descriptores de imagen o descriptores tridimensionales, como muchos otros trabajos aplican. La extracción de la forma y el ajuste de primitivas tridimensionales es un proceso mucho más complejo que la aplicación de descriptores, pero con resultados sobresalientes. Además, los bancos de datos no son necesariamente exhaustivos como es el caso de los descriptores, ya que sólo se necesitan diferentes vistas de un mismo objeto y un proceso de normalización para tener bien caracterizado al mismo. En la Figura 7.4 es posible ver los pasos necesarios para la caracterización de un objeto. Sin ser exhaustivo, el objeto queda bien caracterizado.

El presente trabajo resolvió a cabalidad el problema de la invariancia para la clasificación de objetos hechos por el ser humano. Si bien los resultados de la clasificación son prometedores, es claro que se llegará a un tamaño de banco de datos donde existan objetos similares entre sí por su forma, por lo que esta técnica sería sobrepasada. Sin embargo, se podrían crear enfoques híbridos, en los que la separación de la forma y la descripción de la textura ayuden a crecer el banco de datos sin perder asertividad. Esto será descrito en la Sección 8.3.

Al mejorar la clasificación y reconstrucción de objetos, nuestra propuesta pretende ser una herramienta efectiva en aplicaciones reales, como la navegación automática tanto en espacios *indoor* como *outdoor*, visión robótica, clasificación de ambientes y toma de decisiones por parte de actuadores robóticos.

8.3 Trabajo futuro

El presente trabajo presenta por sí mismo un avance en los procesos de clasificación de objetos hechos por el ser humano, al utilizar los parámetros de las ecuaciones de las primitivas tridimensionales que forman un objeto como atributos de entrada

para el vector característico. Con esto, se obtuvieron mejores resultados que los del estado del arte, como se muestra la Tabla 7.7. Sin embargo, el proceso para extraer los parámetros de las primitivas tridimensionales, que forman un objeto hecho por el ser humano (ver Capítulo 6), aun resulta muy tardado, teniendo un tiempo de ejecución de entre 1.8 a 3.2 segundos en una computadora con procesador Intel Core i7 y 8 GB de RAM. Aunque este tiempo es aceptable respecto a los predecesores en el estado del arte, es aún difícil de manejar por sistemas de navegación automática, robots o sistemas de visión por computadora, siendo el costo a pagar por una identificación más precisa del objeto. La afirmación anterior se traduce en que uno de los retos a futuro debe ser la mejora en los tiempos de ejecución del algoritmo para poder ser usado en problemas reales de una forma eficaz, sin perder asertividad. Esto se puede lograr ya sea mediante programación paralela, el uso de GPUs o la mejora de los procedimientos presentados en el Capítulo 6.

Al incrementar el número de clases identificables por nuestro sistema, se deberá estudiar la posibilidad de incrementar el número de primitivas tridimensionales. Si bien puede parecer una buena idea de inicio, el estudio realizado en la Sección 6.3 advierte que el uso de nuevas primitivas no necesariamente mejorará la asertividad en la clasificación. Además, en la Sección **??** se objeta que la inclusión de más primitivas tridimensionales podría traer consigo repercusiones directas en la complejidad computacional de nuestra propuesta.

El uso de nuevas restricciones geosemánticas es un tema que deberá ser estudiado con cuidado, mediante el análisis del banco de datos. La inclusión de restricciones como ángulos de 45°, simetrías y equidistancias ayudaría a formar los objetos de mejor manera, pero tienen un costo computacional y no en todos los casos es necesario.

El tamaño del banco de datos es una de las mejoras obvias del sistema. El número de clases que pueden ser clasificadas debe crecer para mostrar el alcance de nuestra metodología, así como compararse con trabajos que se dedican a grandes bancos de datos, i.e., trabajos que clasifican más de 250 objetos.

Es claro que los resultados en el proceso de clasificación decaerán al incrementar el número de clases clasificables. Esto se debe a que, en algún momento, no habrá distinción entre una taza y un frasco del mismo tamaño, o entre una caja de regalo y un CPU de similares proporciones. En otras palabras, cada vez habrá más objetos de similar forma y tamaño, lo que dificultaría su correcta clasificación. La forma de solucionar el problema de similaridad (al menos la que resulta obvia en este momento) es la de crear un enfoque probabilístico, resultado de la combinación de técnicas de reconocimiento de objetos para imágenes RGB, i.e., aquellas que hacen uso de descriptores para identificar los objetos de una porción segmentada de la imagen, junto con el presente trabajo, de forma que cada enfoque aporte una probabilidad de ocurrencia según la forma y textura. Siguiendo con el ejemplo de este párrafo, si se quiere identificar entre una caja y un CPU, el estudio del presente trabajo entregará una probabilidad muy similar entre ambos, pero el análisis de texturas podría ayudar a discernir entre ambas, ayudando a incrementar el número de clases y la asertividad.

Los algoritmos desarrollados en este trabajo son invariantes ante el sensor utilizado, por lo que una buena practica sería el uso de otros sensores como Leap Motion para hacer la reconstrucción y probar que la asertividad no depende de los sensores utilizados.

Bibliografía

[Adanja et al., 2010] Adanja, I., Debeir, O., Mégalizzi, V., Kiss, R., Warzée, N., and Decaestecker, C. (2010). Automated tracking of unmarked cells migrating in three-dimensional matrices applied to anti-cancer drug screening. *Experimental cell research*, 316(2):181–193.

[Akl and Toussaint, 1978] Akl, S. G. and Toussaint, G. T. (1978). A fast convex hull algorithm. *Information processing letters*, 7(5):219–222.

[Andrew, 1979] Andrew, A. M. (1979). Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters*, 9(5):216–219.

[Aono and Iwabuchi, 2012] Aono, M. and Iwabuchi, H. (2012). 3d shape retrieval from a 2d image as query. In *Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific*, pages 1–10. IEEE.

[Arbelaez et al., 2011] Arbelaez, P., Maire, M., Fowlkes, C., and Malik, J. (2011). Contour detection and hierarchical image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 33(5):898–916.

[Arcavi, 2003] Arcavi, A. (2003). The role of visual representations in the learning of mathematics. *Educational studies in mathematics*, 52(3):215–241.

[Arman and Aggarwal, 1993] Arman, F. and Aggarwal, J. K. (1993). Model-based object recognition in dense-range images—a review. *Acm Computing Surveys (CSUR)*, 25(1):5–43.

[Arvind et al., 2017] Arvind, V., Costa, A., Badgeley, M., Cho, S., and Oermann, E. (2017). Wide and deep volumetric residual networks for volumetric image classification. *arXiv preprint arXiv:1710.01217*.

[Asghari and Jalali, 2014] Asghari, M. H. and Jalali, B. (2014). Physics-inspired image edge detection. In *Signal and Information Processing (GlobalSIP), 2014 IEEE Global Conference on*, pages 293–296. IEEE.

[Ask et al., 2014] Ask, E., Enqvist, O., Svärm, L., Kahl, F., and Lippolis, G. (2014). Tractable and reliable registration of 2d point sets. In *European Conference on Computer Vision*, pages 393–406. Springer.

[Bae et al., 2008] Bae, S.-H., Balakrishnan, R., and Singh, K. (2008). Ilovesketch: as-natural-as-possible sketching system for creating 3d curve models. In *Proceedings*

of the 21st annual ACM symposium on User interface software and technology, pages 151–160. ACM.

[Ballard, 1981] Ballard, D. H. (1981). Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, 13(2):111–122.

[Barber et al., 1996] Barber, C. B., Dobkin, D. P., and Huhdanpaa, H. (1996). The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483.

[Bay et al., 2006] Bay, H., Tuytelaars, T., and Van Gool, L. (2006). Surf: Speeded up robust features. *Computer vision–ECCV 2006*, pages 404–417.

[Bell et al., 2016] Bell, S., Lawrence Zitnick, C., Bala, K., and Girshick, R. (2016). Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2874–2883.

[Belongie et al., 2002] Belongie, S., Malik, J., and Puzicha, J. (2002). Shape matching and object recognition using shape contexts. *IEEE transactions on pattern analysis and machine intelligence*, 24(4):509–522.

[Beltrán and Mendoza, 2011] Beltrán, A. and Mendoza, S. (2011). Efficient algorithm for real-time handwritten character recognition in mobile devices. In *Electrical Engineering Computing Science and Automatic Control (CCE), 2011 8th International Conference on*, pages 1–6. IEEE.

[Ben-Shabat et al., 2018] Ben-Shabat, Y., Lindenbaum, M., and Fischer, A. (2018). 3dmfv: 3d point cloud classification in real-time using convolutional neural network. *IEEE Robotics and Automation Letters*.

[Bhatt et al., 2010] Bhatt, H. S., Bharadwaj, S., Singh, R., and Vatsa, M. (2010). On matching sketches with digital face images. In *Biometrics: Theory Applications and Systems (BTAS), 2010 Fourth IEEE International Conference on*, pages 1–7. IEEE.

[Black, 1998] Black, P. E. (1998). Dictionary of algorithms and data structures. Technical report.

[Bloomenthal et al., 1998] Bloomenthal, M., Zeleznik, R., Fish, R., Holden, L., Forsberg, A., Riesenfeld, R., Cutts, M., Drake, S., Fuchs, H., and Cohen, E. (1998). Sketch-n-make: Automated machining of cad sketches. In *Proceedings of ASME DETC*, volume 98, pages 1–11.

[Bolles and Fischler, 1981] Bolles, R. C. and Fischler, M. A. (1981). A ransac-based approach to model fitting and its application to finding cylinders in range data. In *IJCAI*, volume 1981, pages 637–643.

[Borrmann et al., 2011] Borrmann, D., Elseberg, J., Lingemann, K., and Nüchter, A. (2011). The 3d hough transform for plane detection in point clouds: A review and a new accumulator design. *3D Research*, 2(2):3.

[Bosché, 2010] Bosché, F. (2010). Automated recognition of 3d cad model objects in laser scans and calculation of as-built dimensions for dimensional compliance control in construction. *Advanced engineering informatics*, 24(1):107–118.

[Bouwmans et al., 2008] Bouwmans, T., El Baf, F., and Vachon, B. (2008). Background modeling using mixture of gaussians for foreground detection-a survey. *Recent Patents on Computer Science*, 1(3):219–237.

[Bravo et al., 2010] Bravo, I., Mazo, M., Lázaro, J. L., Gardel, A., Jiménez, P., and Pizarro, D. (2010). An intelligent architecture based on field programmable gate arrays designed to detect moving objects by using principal component analysis. *Sensors*, 10(10):9232–9251.

[Bulgerin, 2014] Bulgerin, T. (2014). *3D Sketch Recognition Using The Microsoft Kinect*. PhD thesis, Texas State University.

[Canny, 1986] Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698.

[Cao et al., 2013] Cao, X., Zhang, H., Liu, S., Guo, X., and Lin, L. (2013). Sym-fish: A symmetry-aware flip invariant sketch histogram shape descriptor. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 313–320.

[Chan, 1996] Chan, T. M. (1996). Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16(4):361–368.

[Chand and Kapur, 1970] Chand, D. R. and Kapur, S. S. (1970). An algorithm for convex polytopes. *Journal of the ACM (JACM)*, 17(1):78–86.

[Chaouch and Verroust-Blondet, 2007] Chaouch, M. and Verroust-Blondet, A. (2007). A new descriptor for 2d depth image indexing and 3d model retrieval. In *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, volume 6, pages VI–373. IEEE.

[Chazelle, 1982] Chazelle, B. (1982). A theorem on polygon cutting with applications. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 339–349. IEEE.

[Chazelle and Dobkin, 1985] Chazelle, B. and Dobkin, D. P. (1985). Optimal convex decompositions. *Computational geometry*, 4(5):63–133.

[Chen et al., 2015] Chen, K., Lai, Y.-K., and Hu, S.-M. (2015). 3d indoor scene modeling from rgb-d data: a survey. *Computational Visual Media*, 1(4):267–278.

[Chen et al., 2016] Chen, T., Zhu, Z., Hu, S.-M., Cohen-Or, D., and Shamir, A. (2016). Extracting 3d objects from photographs using 3-sweep. *Communications of the ACM*, 59(12):121–129.

[Chen et al., 2013] Chen, T., Zhu, Z., Shamir, A., Hu, S.-M., and Cohen-Or, D. (2013). 3-sweep: Extracting editable objects from a single photo. *ACM Transactions on Graphics (TOG)*, 32(6):195.

[Chen et al., 2008] Chen, X., Kang, S. B., Xu, Y.-Q., Dorsey, J., and Shum, H.-Y. (2008). Sketching reality: Realistic interpretation of architectural designs. *ACM Transactions on Graphics (TOG)*, 27(2):11.

[Choo et al., 2014] Choo, B., Landau, M., DeVore, M., and Beling, P. A. (2014). Statistical analysis-based error models for the microsoft kinecttm depth sensor. *Sensors*, 14(9):17430–17450.

[Claesen et al., 2014] Claesen, M., De Smet, F., Suykens, J. A., and De Moor, B. (2014). Fast prediction with svm models containing rbf kernels. *arXiv preprint arXiv:1403.0736*.

[Company et al., 2005] Company, P., Piquer, A., Contero, M., and Naya, F. (2005). A survey on geometrical reconstruction as a core technology to sketch-based modeling. *Computers & Graphics*, 29(6):892–904.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

[Crabb et al., 2008] Crabb, R., Tracey, C., Puranik, A., and Davis, J. (2008). Real-time foreground segmentation via range and color imaging. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*, pages 1–5. IEEE.

[Cristani et al., 2010] Cristani, M., Farenzena, M., Bloisi, D., and Murino, V. (2010). Background subtraction for automated multisensor surveillance: a comprehensive review. *EURASIP Journal on Advances in signal Processing*, 2010(1):343057.

[Dalal and Triggs, 2005] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE.

[De Berg et al., 2008] De Berg, M., Cheong, O., Van Kreveld, M., and Overmars, M. (2008). *Computational Geometry: Introduction*. Springer.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee.

[D'Hondt et al., 2013] D'Hondt, O., Guillaso, S., and Hellwich, O. (2013). Geometric primitive extraction for 3d reconstruction of urban areas from tomographic sar data. In *Urban Remote Sensing Event (JURSE), 2013 Joint*, pages 206–209. IEEE.

[Dokládal and Dokladalova, 2011] Dokládal, P. and Dokladalova, E. (2011). Computationally efficient, one-pass algorithm for morphological filters. *Journal of Visual Communication and Image Representation*, 22(5):411–420.

[Duda, 1973] Duda, R. O., . H. P. E. (1973). *Pattern classification and scene analysis*. New York: John Wiley and Sons.

[Duda and Hart, 1972] Duda, R. O. and Hart, P. E. (1972). Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15.

[Dudani et al., 1977] Dudani, S. A., Breeding, K. J., and McGhee, R. B. (1977). Aircraft identification by moment invariants. *IEEE transactions on computers*, 100(1):39–46.

[Eggli et al., 1997] Eggli, L., Hsu, C.-y., Brüderlin, B. D., and Elber, G. (1997). Inferring 3d models from freehand sketches and constraints. *Computer-Aided Design*, 29(2):101–112.

[Eitz et al., 2012] Eitz, M., Hays, J., and Alexa, M. (2012). How do humans sketch objects? *ACM Trans. Graph.*, 31(4):44–1.

[Eitz et al., 2011] Eitz, M., Hildebrand, K., Boubekeur, T., and Alexa, M. (2011). Sketch-based image retrieval: Benchmark and bag-of-features descriptors. *IEEE transactions on visualization and computer graphics*, 17(11):1624–1636.

[Elhabian et al., 2008] Elhabian, S. Y., El-Sayed, K. M., and Ahmed, S. H. (2008). Moving object detection in spatial domain using background removal techniques-state-of-art. *Recent patents on computer science*, 1(1):32–54.

[Ellis et al., 1969] Ellis, T., Heafner, J. F., and Sibley, W. (1969). The grail project: An experiment in man-machine communications. Technical report, DTIC Document.

[Fernandez-Sanchez et al., 2013] Fernandez-Sanchez, E. J., Diaz, J., and Ros, E. (2013). Background subtraction based on color and depth using active sensors. *Sensors*, 13(7):8895–8915.

[Figueiredo et al., 2017] Figueiredo, R., Moreno, P., and Bernardino, A. (2017). Automatic object shape completion from 3d point clouds for object manipulation.

[Fischler and Bolles, 1981] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395.

[Gal et al., 2009] Gal, R., Sorkine, O., Mitra, N. J., and Cohen-Or, D. (2009). iwires: an analyze-and-edit approach to shape manipulation. In *ACM Transactions on Graphics (TOG)*, volume 28, page 33. ACM.

[Georgiev et al., 2016] Georgiev, K., Al-Hami, M., and Lakaemper, R. (2016). Real-time 3d scene description using spheres, cones and cylinders. *arXiv preprint arXiv:1603.03856*.

[Gingold et al., 2009] Gingold, Y., Igarashi, T., and Zorin, D. (2009). Structured annotations for 2d-to-3d modeling. In *ACM Transactions on Graphics (TOG)*, volume 28, page 148. ACM.

[Graham, 1972] Graham, R. L. (1972). An efficient algorith for determining the convex hull of a finite planar set. *Information processing letters*, 1(4):132–133.

[Guo et al., 2007] Guo, C.-e., Zhu, S.-C., and Wu, Y. N. (2007). Primal sketch: Integrating structure and texture. *Computer Vision and Image Understanding*, 106(1):5–19.

[Ha and Eck, 2017] Ha, D. and Eck, D. (2017). A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*.

[Harris and Stephens, 1988] Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Manchester, UK.

[Hartley and Zisserman, 2003] Hartley, R. and Zisserman, A. (2003). *Multiple view geometry in computer vision*. Cambridge university press.

[He and Sun, 2015] He, K. and Sun, J. (2015). Convolutional neural networks at constrained time cost. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5353–5360.

[Hernández-Vela et al., 2012] Hernández-Vela, A., Reyes, M., Ponce, V., and Escalera, S. (2012). Grabcut-based human segmentation in video sequences. *Sensors*, 12(11):15376–15393.

[Hu and Collomosse, 2013] Hu, R. and Collomosse, J. (2013). A performance evaluation of gradient field hog descriptor for sketch based image retrieval. *Computer Vision and Image Understanding*, 117(7):790–806.

[Huber et al., 2004] Huber, D., Kapuria, A., Donamukkala, R., and Hebert, M. (2004). Parts-based 3d object classification. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–II. IEEE.

[Ibrahim and Rahimian, 2010] Ibrahim, R. and Rahimian, F. P. (2010). Comparison of cad and manual sketching tools for teaching architectural design. *Automation in Construction*, 19(8):978–987.

[Inglada, 2007] Inglada, J. (2007). Automatic recognition of man-made objects in high resolution optical remote sensing images by svm classification of geometric image features. *ISPRS journal of photogrammetry and remote sensing*, 62(3):236–248.

[Ip et al., 2002] Ip, C. Y., Lapadat, D., Sieger, L., and Regli, W. C. (2002). Using shape distributions to compare solid models. In *Proceedings of the seventh ACM symposium on Solid modeling and applications*, pages 273–280. ACM.

[J. Ostro et al., 2002] J. Ostro, S., Hudson, S., A. M. Benner, L., D. Giorgini, J., Magri, C., L. Margot, J., and Nolan, M. (2002). Asteroid radar astronomy.

[Jiménez-Hernández, 2010] Jiménez-Hernández, H. (2010). Background subtraction approach based on independent component analysis. *Sensors*, 10(6):6092–6114.

[Joachimiak et al., 2013] Joachimiak, M., Hannuksela, M. M., and Gabbouj, M. (2013). Complete processing chain for 3d video generation using kinect sensor. In *3D Imaging (IC3D), 2013 International Conference on*, pages 1–7. IEEE.

[Joaquim and Jorge, 2004] Joaquim, M. J. F. A. F. and Jorge, A. (2004). Towards 3d modeling using sketches and retrieval. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling 2004*, page 127. Citeseer.

[Johnson and Hebert, 1999] Johnson, A. E. and Hebert, M. (1999). Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (5):433–449.

[Johnson et al., 2009] Johnson, G., Gross, M. D., Hong, J., Do, E. Y.-L., et al. (2009). Computational support for sketching in design: a review. *Foundations and Trends® in Human–Computer Interaction*, 2(1):1–93.

[Johnson, 1963] Johnson, T. E. (1963). Sketchpad iii: a computer program for drawing in three dimensions. In *Proceedings of the May 21-23, 1963, spring joint computer conference*, pages 347–353. ACM.

[Jorge et al., 2003] Jorge, J. A., Silva, N. F., and Cardoso, T. D. (2003). Gides+. *In: Proceedings of the 12th annual Portuguese CG meeting*.

[Keil and Snoeyink, 2002] Keil, M. and Snoeyink, J. (2002). On the time bound for convex decomposition of simple polygons. *International Journal of Computational Geometry & Applications*, 12(03):181–192.

[Kirkpatrick and Seidel, 1986] Kirkpatrick, D. G. and Seidel, R. (1986). The ultimate planar convex hull algorithm? *SIAM journal on computing*, 15(1):287–299.

[Kowsari and Alassaf, 2016] Kowsari, K. and Alassaf, M. H. (2016). Weighted unsupervised learning for 3d object detection. *arXiv preprint arXiv:1602.05920*.

[Kratt et al., 2014] Kratt, J., Eisenkeil, F., Pirk, S., Sharf, A., and Deussen, O. (2014). Non-realistic 3d object stylization. In *Proceedings of the Workshop on Computational Aesthetics*, pages 67–75. ACM.

[Lacey et al., 2000] Lacey, A., Pinitkarn, N., and Thacker, N. A. (2000). An evaluation of the performance of ransac algorithms for stereo camera calibrarion. In *BMVC*, pages 1–10.

[Lau et al., 2011] Lau, M., Ohgawara, A., Mitani, J., and Igarashi, T. (2011). Converting 3d furniture models to fabricatable parts and connectors. In *ACM Transactions on Graphics (TOG)*, volume 30, page 85. ACM.

[Lau et al., 2010] Lau, M., Saul, G., Mitani, J., and Igarashi, T. (2010). Modeling-in-context: user design of complementary objects with a single photo. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*, pages 17–24. Eurographics Association.

[Le and Duan, 2017] Le, T. and Duan, Y. (2017). A primitive-based 3d segmentation algorithm for mechanical cad models. *Computer Aided Geometric Design*, 52:231–246.

[Lee and Park, 2012] Lee, J. and Park, M. (2012). An adaptive background subtraction method based on kernel density estimation. *Sensors*, 12(9):12279–12300.

[Lendaris and Stanley, 1970] Lendaris, G. G. and Stanley, G. L. (1970). Diffraction-pattern sampling for automatic pattern recognition. *Proceedings of the IEEE*, 58(2):198–216.

[Lenz et al., 2015] Lenz, I., Lee, H., and Saxena, A. (2015). Deep learning for detecting robotic grasps. *The International Journal of Robotics Research*, 34(4-5):705–724.

[Li and Johan, 2013] Li, B. and Johan, H. (2013). Sketch-based 3d model retrieval by incorporating 2d-3d alignment. *Multimedia Tools and Applications*, 65(3):363–385.

[Li et al., 2015a] Li, B., Lu, Y., Ghumman, A., Strylowski, B., Gutierrez, M., Sadiq, S., Forster, S., Feola, N., and Bugerin, T. (2015a). Kinectsbr: A kinect-assisted 3d sketch-based 3d model retrieval system. In *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, pages 655–656. ACM.

[Li et al., 2003] Li, L., Huang, W., Gu, I. Y., and Tian, Q. (2003). Foreground object detection from videos containing complex background. In *Proceedings of the eleventh ACM international conference on Multimedia*, pages 2–10. ACM.

[Li et al., 2015b] Li, Y., Hospedales, T. M., Song, Y.-Z., and Gong, S. (2015b). Free-hand sketch recognition by multi-kernel feature learning. *Computer Vision and Image Understanding*, 137:1–11.

[Li et al., 2013] Li, Y., Song, Y.-Z., and Gong, S. (2013). Sketch recognition by ensemble matching of structured features. In *BMVC*, volume 1, page 2.

[Li et al., 2011] Li, Y., Wu, X., Chrysathou, Y., Sharf, A., Cohen-Or, D., and Mitra, N. J. (2011). Globfit: Consistently fitting primitives by discovering global relations. In *ACM Transactions on Graphics (TOG)*, volume 30, page 52. ACM.

[Lien and Amato, 2006] Lien, J.-M. and Amato, N. M. (2006). Approximate convex decomposition of polygons. *Computational Geometry*, 35(1-2):100–123.

[Lindeberg, 1996] Lindeberg, T. (1996). Edge detection and ridge detection with automatic scale selection. In *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR'96, 1996 IEEE Computer Society Conference on*, pages 465–470. IEEE.

[Lingas, 1982] Lingas, A. (1982). The power of non-rectilinear holes. In *International Colloquium on Automata, Languages, and Programming*, pages 369–383. Springer.

[Liu and Chen, 2007] Liu, G.-h. and Chen, C.-b. (2007). A new algorithm for computing the convex hull of a planar point set. *Journal of Zhejiang University-SCIENCE A*, 8(8):1210–1217.

[Liu et al., 2010] Liu, H., Liu, W., and Latecki, L. J. (2010). Convex shape decomposition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 97–104. IEEE.

[Liu et al., 2012] Liu, R., Fang, B., Tang, Y. Y., Wen, J., and Qian, J. (2012). A fast convex hull algorithm with maximum inscribed circle affine transformation. *Neurocomputing*, 77(1):212–221.

[Louppe, 2014] Louppe, G. (2014). Understanding random forests: From theory to practice. *arXiv preprint arXiv:1407.7502*.

[Lowe, 1999] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee.

[MacQueen et al., 1967] MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA.

[Malik et al., 2016] Malik, J., Arbeláez, P., Carreira, J., Fragkiadaki, K., Girshick, R., Gkioxari, G., Gupta, S., Hariharan, B., Kar, A., and Tulsiani, S. (2016). The three r's of computer vision: Recognition, reconstruction and reorganization. *Pattern Recognition Letters*, 72:4–14.

[Marr and Hildreth, 1980] Marr, D. and Hildreth, E. (1980). Theory of edge detection. *Proceedings of the Royal Society of London B: Biological Sciences*, 207(1167):187–217.

[Meng et al., 2013] Meng, M., Xia, J., Luo, J., and He, Y. (2013). Unsupervised co-segmentation for 3d shapes using iterative multi-label optimization. *Computer-Aided Design*, 45(2):312–320.

[Messinger et al., 2010] Messinger, D., Ziemann, A., Schlamm, A., and Basener, B. (2010). Spectral image complexity estimated through local convex hull volume. In *Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS), 2010 2nd Workshop on*, pages 1–4. IEEE.

[Morel and Yu, 2009] Morel, J.-M. and Yu, G. (2009). Asift: A new framework for fully affine invariant image comparison. *SIAM Journal on Imaging Sciences*, 2(2):438–469.

[Mosig et al., 2009] Mosig, A., Jäger, S., Wang, C., Nath, S., Ersoy, I., Palaniappan, K.-p., and Chen, S.-S. (2009). Tracking cells in life cell imaging videos using topological alignments. *Algorithms for Molecular Biology*, 4(1):10.

[Nocedal and Wright, 2006] Nocedal, J. and Wright, S. J. (2006). Numerical optimization 2nd.

[Oehler et al., 2011] Oehler, B., Stueckler, J., Welle, J., Schulz, D., and Behnke, S. (2011). Efficient multi-resolution plane segmentation of 3d point clouds. In *International Conference on Intelligent Robotics and Applications*, pages 145–156. Springer.

[Olsen et al., 2009] Olsen, L., Samavati, F. F., Sousa, M. C., and Jorge, J. A. (2009). Sketch-based modeling: A survey. *Computers & Graphics*, 33(1):85–103.

[Osada et al., 2001] Osada, R., Funkhouser, T., Chazelle, B., and Dobkin, D. (2001). Matching 3d models with shape distributions. In *Shape Modeling and Applications, SMI 2001 International Conference On.*, pages 154–166. IEEE.

[Otsu, 1979] Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66.

[Pang et al., 2015] Pang, G., Qiu, R., Huang, J., You, S., Neumann, U., et al. (2015). Automatic 3d industrial point cloud classification and modeling. In *SPE Western Regional Meeting*. Society of Petroleum Engineers.

[Paul, 1962] Paul, H. (1962). Method and means for recognizing complex patterns. US Patent 3,069,654.

[Paulson and Hammond, 2008] Paulson, B. and Hammond, T. (2008). Paleosketch: accurate primitive sketch recognition and beautification. In *Proceedings of the 13th international conference on Intelligent user interfaces*, pages 1–10. ACM.

[Plumed et al., 2013] Plumed, R., Varley, P. A. C., et al. (2013). Human-like recognition of straight lines in sketched strokes.

[Pluskal et al., 2010] Pluskal, T., Castillo, S., Villar-Briones, A., and Orešič, M. (2010). Mzmine 2: modular framework for processing, visualizing, and analyzing mass spectrometry-based molecular profile data. *BMC bioinformatics*, 11(1):395.

[Prewitt, 1970] Prewitt, J. M. (1970). Object enhancement and extraction. *Picture processing and Psychopictorics*, 10(1):15–19.

[Pugh, 1992] Pugh, D. (1992). Designing solid objects using interactive sketch interpretation. In *Proceedings of the 1992 symposium on Interactive 3D graphics*, pages 117–126. ACM.

[Qi et al., 2017] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4.

[Qi et al., 2016] Qi, C. R., Su, H., Nießner, M., Dai, A., Yan, M., and Guibas, L. J. (2016). Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5648–5656.

[Rangel et al., 2018] Rangel, J. C., Martínez-Gómez, J., Romero-González, C., García-Varea, I., and Cazorla, M. (2018). Semi-supervised 3d object recognition through cnn labeling. *Applied Soft Computing*, 65:603–613.

[Redmon and Angelova, 2015] Redmon, J. and Angelova, A. (2015). Real-time grasp detection using convolutional neural networks. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1316–1322. IEEE.

[Ren et al., 2011] Ren, Z., Yuan, J., Li, C., and Liu, W. (2011). Minimum near-convex decomposition for robust shape representation. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 303–310. IEEE.

[Ren et al., 2013] Ren, Z., Yuan, J., and Liu, W. (2013). Minimum near-convex shape decomposition. *IEEE transactions on pattern analysis and machine intelligence*, 35(10):2546–2552.

[Renwu et al., 2011] Renwu, L., Dehong, Y., and Yan, L. (2011). and chen ke2)(1) college of land resource engineering, kunming university of sicence and technology, kunming 6500932) college of land and resource, china west normal university, nanchong 637008); an improved algorithm for producing minimum convex hull [j]. *Journal of Geodesy and Geodynamics*, 3.

[Rodriguez-Gomez et al., 2015] Rodriguez-Gomez, R., Fernandez-Sanchez, E. J., Diaz, J., and Ros, E. (2015). Codebook hardware implementation on fpga for background subtraction. *Journal of Real-Time Image Processing*, 10(1):43–57.

[Rosten et al., 2010] Rosten, E., Porter, R., and Drummond, T. (2010). Faster and better: A machine learning approach to corner detection. *IEEE transactions on pattern analysis and machine intelligence*, 32(1):105–119.

[Ruiz et al., 2014] Ruiz, C. R., Le, S. N., Yu, J., and Low, K.-L. (2014). Multi-style paper pop-up designs from 3d models. In *Computer Graphics Forum*, volume 33, pages 487–496. Wiley Online Library.

[Russell et al., 2008] Russell, B. C., Torralba, A., Murphy, K. P., and Freeman, W. T. (2008). Labelme: a database and web-based tool for image annotation. *International journal of computer vision*, 77(1):157–173.

[Rusu et al., 2009] Rusu, R. B., Blodow, N., and Beetz, M. (2009). Fast point feature histograms (fpfh) for 3d registration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3212–3217. Citeseer.

[Ryoo and Aggarwal, 2009] Ryoo, M. S. and Aggarwal, J. K. (2009). Spatio-temporal relationship match: Video structure comparison for recognition of complex human activities. In *Computer vision, 2009 ieee 12th international conference on*, pages 1593–1600. IEEE.

[Schiller and Koch, 2011] Schiller, I. and Koch, R. (2011). Improved video segmentation by adaptive combination of depth keying and mixture-of-gaussians. In *Scandinavian conference on Image analysis*, pages 59–68. Springer.

[Schneider and Tuytelaars, 2014] Schneider, R. G. and Tuytelaars, T. (2014). Sketch classification and classification-driven analysis using fisher vectors. *ACM Transactions on Graphics (TOG)*, 33(6):174.

[Schwarz et al., 2015] Schwarz, M., Schulz, H., and Behnke, S. (2015). Rgb-d object recognition and pose estimation based on pre-trained convolutional neural network features. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1329–1335. IEEE.

[Scovanner et al., 2007] Scovanner, P., Ali, S., and Shah, M. (2007). A 3-dimensional sift descriptor and its application to action recognition. In *Proceedings of the 15th ACM international conference on Multimedia*, pages 357–360. ACM.

[Serna and Marcotegui, 2014] Serna, A. and Marcotegui, B. (2014). Detection, segmentation and classification of 3d urban objects using mathematical morphology

and supervised learning. *ISPRS Journal of Photogrammetry and Remote Sensing*, 93:243–255.

[Shao et al., 2012] Shao, T., Xu, W., Zhou, K., Wang, J., Li, D., and Guo, B. (2012). An interactive approach to semantic modeling of indoor scenes with an rgbd camera. *ACM Transactions on Graphics (TOG)*, 31(6):136.

[Sharif et al., 2011] Sharif, M., Naqvi, S. Z. Z., Raza, M., and Haider, W. (2011). A new approach to compute convex hull. *Innov. Syst. Design Eng*, 2(3):186–192.

[Shin and Igarashi, 2007] Shin, H. and Igarashi, T. (2007). Magic canvas: interactive design of a 3-d scene prototype from freehand sketches. In *Proceedings of Graphics Interface 2007*, pages 63–70. ACM.

[Shpitalni and Lipson, 1997] Shpitalni, M. and Lipson, H. (1997). Classification of sketch strokes and corner detection using conic sections and adaptive clustering. *Transactions of the ASME-R-Journal of Mechanical Design*, 119(1):131–134.

[Shtof et al., 2013] Shtof, A., Agathos, A., Gingold, Y., Shamir, A., and Cohen-Or, D. (2013). Geosemantic snapping for sketch-based modeling. In *Computer Graphics Forum*, volume 32, pages 245–253. Wiley Online Library.

[Singh and Singh, 2013] Singh, N. H. and Singh, L. D. (2013). Faster convex hull computation by reducing computatinal overhead and time. *International Journal of Scientific Engineering Research*, 4(12):454–460.

[Sivic et al., 2003] Sivic, J., Zisserman, A., et al. (2003). Video google: A text retrieval approach to object matching in videos. In *iccv*, volume 2, pages 1470–1477.

[Socher et al., 2012] Socher, R., Huval, B., Bath, B., Manning, C. D., and Ng, A. Y. (2012). Convolutional-recursive deep learning for 3d object classification. In *Advances in neural information processing systems*, pages 656–664.

[Song and Xiao, 2014] Song, S. and Xiao, J. (2014). Sliding shapes for 3d object detection in depth images. In *European conference on computer vision*, pages 634–651. Springer.

[Souviron, 2013] Souviron, J. (2013). Convex hull: Incremental variations on the akl-toussaint heuristics simple, optimal and space-saving convex hull algorithms. *arXiv preprint arXiv:1304.2676*.

[Staranowicz et al., 2013] Staranowicz, A., Brown, G. R., Morbidi, F., and Mariottini, G. L. (2013). Easy-to-use and accurate calibration of rgb-d cameras from spheres. In *Pacific-Rim Symposium on Image and Video Technology*, pages 265–278. Springer.

[Steder et al., 2010] Steder, B., Rusu, R. B., Konolige, K., and Burgard, W. (2010). Narf: 3d range image features for object recognition. In *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, volume 44.

[Sutherland, 1964] Sutherland, I. E. (1964). Sketchpad a man-machine graphical communication system. *Transactions of the Society for Computer Simulation*, 2(5):R–3.

[Taele and Hammond, 2014] Taele, P. and Hammond, T. (2014). Developing sketch recognition and interaction techniques for intelligent surfaceless sketching user interfaces. In *Proceedings of the companion publication of the 19th international conference on Intelligent User Interfaces*, pages 53–56. ACM.

[Taillandier et al., 2010] Taillandier, P., Vo, D.-A., Amouroux, E., and Drogoul, A. (2010). Gama: a simulation platform that integrates geographical information data, agent-based modeling and multi-scale control. In *International Conference on Principles and Practice of Multi-Agent Systems*, pages 242–258. Springer.

[Tseng et al., 2012] Tseng, K.-Y., Lin, Y.-L., Chen, Y.-H., and Hsu, W. H. (2012). Sketch-based image retrieval on mobile devices using compact hash bits. In *Proceedings of the 20th ACM international conference on Multimedia*, pages 913–916. ACM.

[Tumen et al., 2010] Tumen, R. S., Acer, M. E., and Sezgin, T. M. (2010). Feature extraction and classifier combination for image-based sketch recognition. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*, pages 63–70. Eurographics Association.

[Walther et al., 2011] Walther, D. B., Chai, B., Caddigan, E., Beck, D. M., and Fei-Fei, L. (2011). Simple line drawings suffice for functional mri decoding of natural scene categories. *Proceedings of the National Academy of Sciences*, 108(23):9661–9666.

[Wang et al., 2017] Wang, Y., Xu, H., Yu, G., and Liang, J. (2017). Coarse pose registration using local geometric features. In *International Symposium on Computer Science and Artificial Intelligence (ISCSAI)*, volume 1, pages 45–48.

[Witten et al., 2016] Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. (2016). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

[Wohlkinger and Vincze, 2011] Wohlkinger, W. and Vincze, M. (2011). Ensemble of shape functions for 3d object classification. In *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, pages 2987–2992. IEEE.

[Wu et al., 2015] Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. (2015). 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920.

[Xiangyu et al., 2006] Xiangyu, Y., Hong, S., and Zhixiong, Y. (2006). An improved algorithm to determine the convex hull of 2d points set. *Journal of Wuhan University of Technology*, 27(10):81–83.

[Xu et al., 2011] Xu, K., Zheng, H., Zhang, H., Cohen-Or, D., Liu, L., and Xiong, Y. (2011). Photo-inspired model-driven 3d object modeling. In *ACM Transactions on Graphics (TOG)*, volume 30, page 80. ACM.

[Yang et al., 2005] Yang, C., Sharon, D., and van de Panne, M. (2005). Sketch-based modeling of parameterized objects. In *SIGGRAPH Sketches*, page 89.

[Zeleznik et al., 1996] Zeleznik, R. C., Herndon, K. P., and Hughes, J. F. (1996). Sketch: An interface for sketching 3d scenes. In *ACM SIGGRAPH 1996 courses*, page 19. ACM.

[Zeleznik et al., 2007] Zeleznik, R. C., Herndon, K. P., and Hughes, J. F. (2007). Sketch: An interface for sketching 3d scenes. In *ACM SIGGRAPH 2007 courses*, page 19. ACM.

[Zhang and Zhang, 2011] Zhang, C. and Zhang, Z. (2011). *Calibration between Depth and Color Sensors for Commodity Depth Cameras*, pages 47–64. Springer International Publishing.

[Zhang et al., 2002] Zhang, D., Lu, G., et al. (2002). A comparative study of fourier descriptors for shape representation and retrieval. In *Proc. of 5th Asian Conference on Computer Vision (ACCV)*, pages 646–651. Citeseer.

[Zhang and Chen, 2014] Zhang, L. and Chen, X. (2014). Topology-based automatic 3d modeling from multiple images. In *Wireless Communications and Signal Processing (WCSP), 2014 Sixth International Conference on*, pages 1–6. IEEE.

[Zhang, 2000] Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334.

[Zhou et al., 2014] Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A. (2014). Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*.

[Zhou et al., 2010] Zhou, X., Jiang, W., Tian, Y., and Shi, Y. (2010). Kernel subclass convex hull sample selection method for svm on face recognition. *Neurocomputing*, 73(10):2234–2246.

[Zhu et al., 2009a] Zhu, J., Liao, M., Yang, R., and Pan, Z. (2009a). Joint depth and alpha matte optimization via fusion of stereo and time-of-flight sensor. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 453–460. IEEE.

[Zhu et al., 2009b] Zhu, N., Wang, G., Yang, G., and Dai, W. (2009b). A fast 2d otsu thresholding algorithm based on improved histogram. In *Pattern Recognition, 2009. CCPR 2009. Chinese Conference on*, pages 1–5. IEEE.

[Zou et al., 2017] Zou, C., Yumer, E., Yang, J., Ceylan, D., and Hoiem, D. (2017). 3d-prnn: Generating shape primitives with recurrent neural networks. In *The IEEE International Conference on Computer Vision (ICCV)*.