UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

RECONOCIMIENTO 3D DE OBJETOS UTILIZANDO MODELOS OCULTOS DE MARKOV PARA ROBOTS DE SERVICIO

T E S I S

QUE PARA OPTAR EL GRADO DE:

MAESTRO EN INGENIERÍA (COMPUTACIÓN)

P R E S E N T A:

LUIS ALFREDO JUÁREZ BLANCO

DIRECTOR DE TESIS:
DR. JESÚS SAVAGE CARMONA
FACULTAD DE INGENIERÍA - UNAM
MÉXICO, D.F., ENERO DE 2016

Agradecimientos

A mis hijas Itziar y Jatziri por ser la energía que me impulsa cada día y por ofrecerme los momentos más felices de mi vida.

Agradezco a mis padres por su apoyo incondicional.

Al Dr. Jesús Savage Carmona por darme la oportunidad de ser parte de su excelente grupo de trabajo y a los compañeros del laboratorio de biorobótica por compartir sus conocimientos conmigo.

Gracias a mis revisores, el Dr. Boris Escalante Ramírez, Dr. Francisco Javier García Ugalde, Dr. Fernando Arámbula Cosío y el M. en I. Abel Pacheco Ortega por el tiempo dedicado y la buena orientación durante el desarrollo de este trabajo.

A todos los profesores-investigadores que fueron parte de mi formación, a la UNAM, al IIMAS. En general muchas gracias y una disculpa a todos los afectados en esta excelente aventura.

Se agradece a CONACYT por hacerme llegar los recursos económicos necesarios para el estudio de esta maestría y a la DGAPA-UNAM por el apoyo proporcionado para la realización de esta tesis a través del proyecto PAPIIT IG100915 "Desarrollo de técnicas de la robótica aplicadas a las artes escénicas y visuales"

II

Resumen

En el presente trabajo de tesis se propone una nueva técnica para mejorar el reconocimiento de objetos en los robots Justina y Judy utilizando nubes de puntos y Modelos Ocultos de Markov Discretos. Justina y Judy son dos robots de servicio desarrollados en el laboratorio de Biorobótica de la Facultad de Ingeniería en la UNAM, ambos utilizan en su sistema de visión, algoritmos de reconocimiento basados en descriptores locales, estos algoritmos han demostrado ser ineficientes cuando trabajan con imágenes obtenidas en ambientes con iluminación no controlada o cuando se desea reconocer un objeto por su forma. Debido a los problemas anteriores se propone utilizar la nube de puntos que el dispositivo Kinect proporciona, así como Modelos Ocultos de Markov para calcular la probabilidad máxima de correspondencia de un objeto actualmente observado con los modelos anteriormente entrenados.

Se presentan diferentes técnicas para el pre procesamiento de la nube de puntos con el objetivo de conseguir observaciones fiables y se plantea una manera de obtener estas observaciones utilizando curvaturas discretizadas.

Con esta propuesta se pretende fortalecer el reconocimiento de objetos en condiciones donde el sistema de reconocimiento actual no pueda garantizar buenos resultados.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

INTRODUCCIÓN

1.1. Presentación

El reconocimiento de objetos ha sido siempre un reto para la visión computacional y muchos algoritmos para este fin se han desarrollado, utilizar estos algoritmos de reconocimiento existentes en un robot sobre ambientes no controlados y con hardware de bajo costo resulta en una tarea desafiante que requiere del uso de técnicas específicas para casos específicos de reconocimiento, requiriéndose desde los muy utilizados algoritmos basados en características hasta aquellos que se especializan en formas o colores y que suelen ser ineficientes y muy lentos cuando se trata de reconocimiento en tiempo real y bajo condiciones no controladas en el ambiente. Un problema existente en el área de la robótica surge por la necesidad de reconocer objetos en condiciones altamente variantes de iluminación, con características de información RGB prácticamente nulas e incluso saber diferenciar un objeto real de una imagen, es de esta forma en la cual se hace necesaria información tridimensional que pueda ser adquirida independientemente de la información RGB. Actualmente gracias al surgimiento de dispositivos económicos como el Kinect, que son capaces de proporcionarnos datos para generar una nube de puntos del ambiente, se abre una posibilidad para dar solución a este inconveniente, por tanto, en la presente tesis se busca contribuir en la mejora del reconocimiento con la implementación de una nueva técnica en el robot que utilice la información de profundidad que nos proporciona el Kinect para reconocer objetos cuando se presenten las condiciones críticas arriba mencionadas.

1.2. Objetivo general

Desarrollar un algoritmo que a través de diversas observaciones de la nube de puntos de un objeto que se encuentra sobre un área plana y previo entrenamiento del modelo de muchos otros, nos calcule la máxima probabilidad de que dichas observaciones pertenezcan a uno de los modelos en la base de conocimiento utilizando Modelos Ocultos de Markov Discretos. Implementar el algoritmo anterior en un robot de servicio para buscar optimizar el reconocimiento en situaciones complejas donde los algoritmos actualmente implementados no consiguen los mejores resultados.

1.3. Objetivos específicos

Para lograr el objetivo general antes descrito se necesitan cubrir objetivos más específicos, a continuación se mencionan:

- Estudiar la literatura existente sobre el tema, el uso que se ha dado a los Modelos Ocultos de Markov y las diferentes maneras de caracterizar una nube de puntos.
- Capturar, segmentar y afinar la nube de puntos correspondiente a un objeto localizado en un área plana.
- Obtener observaciones fiables de la nube de puntos utilizando alguna adaptación de descriptores 3D conocidos que mejor se adapte a las necesidades y que permita la generación de un alfabeto bien definido para realizar el entrenamiento.
- Desarrollar un entrenador de objetos para las observaciones obtenidas de la nube de puntos utilizando algoritmos de entrenamiento para Modelos Ocultos de Markov Discretos.
- Desarrollar un sistema de reconocimiento usando Modelos Ocultos de Markov Discretos para calcular la probabilidad de que hechas ciertas observaciones estas correspondan con alguno de los modelos anteriormente entrenados.
- Mejorar el reconocimiento de objetos actual en los robots de servicio Judy y Justina con ayuda del nuevo sistema de reconocimiento propuesto.

1.4. Motivación del trabajo de tesis

La robótica es un área multidisciplinaria que ha ido tomando auge en los últimos años, una de las disciplinas de más importante contribución a la robótica debido a la importancia que representa la interacción humano – máquina es la visión computacional, a través de esta es posible obtener mucha información del ambiente con el cual se interactúa. Debido a este auge, actualmente existen diversos concursos a nivel mundial como RoboCup que buscan motivar a la comunidad académica, científica y público en general para que se involucren en el desarrollo de esta tecnología, es en estos concursos en donde se ponen a prueba las habilidades para resolver los problemas más comunes para los que un robot debería estar preparado, dentro de las habilidades evaluadas está el reconocimiento de objetos, mismo que representa un desafío significativo al tener que enfrentarse a condiciones ambientales altamente cambiantes que alteran la información necesaria para efectuar un reconocimiento aceptable, además de tener que lidiar con diferentes algoritmos que se adapten a las características del objeto a reconocer que la mayoría de las veces resultan lentos e ineficientes.

En el laboratorio de Biorobótica que dirige el Dr. Jesús Savage Carmona en la UNAM, se desarrolla desde hace más de 10 años un proyecto de robótica denominado Robot de Servicio con el cual se ha participado en diferentes concursos nacionales e internacionales. En este proyecto han colaborado estudiantes de diferentes disciplinas y diferentes grados académicos, cada uno contribuyendo en la creación y mejoramiento de las funciones del robot para poder resolver los nuevos desafíos que año con año se presentan en los eventos donde se participa.

Como parte del equipo que trabaja en el Robot de Servicio y tomando en cuenta los problemas del sistema de visión actual, entre ellos los mencionados anteriormente, se busca contribuir con un algoritmo que sea capaz de reconocer objetos de manera eficiente usando la nube de puntos que suministra el dispositivo Kinect, logrando realizar además un rápido reconocimiento que considere la forma del objeto y que sea robusto a condiciones cambiantes del ambiente.

1.5. Hipótesis

El uso de Modelos Ocultos de Markov Discretos sobre una nube de puntos mejorará los resultados del reconocimiento de objetos en el sistema de reconocimiento actualmente implementado en los robots de servicio Judy y Justina, cuando los objetos a reconocer proporcionan más información por forma que por el diseño gráfico en su imagen.

1.6. Estructura de la tesis

El trabajo de tesis que a continuación se presenta, se encuentra estructurado de la siguiente manera: en el capítulo 2 se mencionan los antecedentes y la teoría fundamental para la realización de este trabajo; el capítulo 3 describe la arquitectura del sistema propuesto y el proceso de desarrollo que se ha seguido para la programación de los algoritmos, así como las diferentes técnicas utilizadas para la mejora de los resultados; el capítulo 4 refiere sobre la forma en que se lleva a cabo la implementación de los algoritmos en el sistema de visión de los robots de servicio Judy y Justina para mejorar los resultados del reconocimiento; en el 5° capítulo se hacen pruebas del algoritmo en diversas condiciones y bajo diferentes escenarios, se realizan pruebas del algoritmo de forma independiente y como una mejora del módulo de reconocimiento vigente en el sistema de visión, también se hace una comparación de los resultados que se obtienen antes y después de la implementación; finalmente el capítulo 6 concluye el trabajo e incluye algunas ideas para continuar mejorando a futuro, se agregan también las fuentes consultadas.

Capítulo 2

ANTECEDENTES Y MARCO TEÓRICO

La robótica es una disciplina que ha reunido a muchas otras para complementarse, la visión computacional ha sido una de ellas, misma que se ha integrado como parte importante en lo que respecta a la interacción de la máquina con su entorno. La más temprana mención de un agente artificial guiado visualmente aparece en la mitología clásica[1]. En la comunidad científica y académica existe un gran interés respecto al potencial que representa el campo de la visión computacional y procesamiento de imágenes, la información que se puede extraer por este medio es enorme; la industria privada también ha visto una oportunidad para lograr incrementar su productividad a través de la automatización, utilizando sistemas de visión.

Aunado a lo anterior y junto a los inicios de la robótica y sistemas autónomos, es posible visualizar como se desprende una necesidad por comprender el ambiente que rodea a estos sistemas para tener la información suficiente en un proceso de toma de decisiones. Así, a principio de los 60's nace la visión computacional moderna, la cual se ha encargado de proveer las herramientas necesarias para esta labor [1].

2.1. Reconocimiento

2.1.1. Reconocimiento de objetos

Dentro del proceso de comprensión del ambiente existe una fase de reconocimiento de los diferentes objetos que en este se encuentran, así surgen los

primeros algoritmos dirigidos al reconocimiento. Estos algoritmos se desarrollaron en un principio utilizando técnicas muy básicas para dar solución también a problemas básicos, una de estas técnicas muy frecuentemente utilizadas ha sido poder diferenciar objetos por color, muy práctico en situaciones donde se necesitaba automatizar la clasificación de productos, por poner un ejemplo. Con el tiempo muchas otras técnicas se propusieron con el objetivo de poder extraer la mayor información posible de una imagen, desde umbralización por tonos de gris, detección de bordes, detección de esquinas, detección de formas, descripción a través de histogramas, diferentes tipos de transformadas y muchas más que han incrementado notablemente las capacidades de los sistemas de visión actuales.

Una propiedad importante sobre la mayoría de los algoritmos de reconocimiento de objetos fue propuesta por Roberts a principios de los 60's en su tesis doctoral titulada Machine Perception of Three Dimensional Solids, en el cual identificó la necesidad de relacionar características extraídas de diferentes imágenes con la representación tridimensional de objetos [2]. La propuesta hecha por Roberts marcó un punto de partida para el desarrollo de algoritmos de reconocimiento que hasta la actualidad siguen utilizándose.

La extracción de características y la búsqueda de correspondencias entre ellas para reconocer objetos, no solo se ha utilizado sobre imágenes 2D, esta técnica también se ha aplicado sobre modelos 3D de los objetos, de este modo se hace evidente una necesidad por describir estos modelos, así como los objetos en una escena a través de características extraídas para posteriormente establecer las correspondencias entre ellas. Podemos decir entonces que el trabajo de reconocimiento, ya sea en imágenes o en modelos tridimensionales, se ha centrado en encontrar la mejor estrategia para representar características que los describan (descriptores) y la mejor estrategia para establecer las correspondencias.

En lo que respecta a descriptores para modelos 3D, se han planteado algunos tipos, en [3] se proponen los siguientes: **descriptores de superficie** donde los objetos se clasifican en función de las superficies que representan; **de curvas y rectas** donde los objetos se describen en base a los contornos o discontinuidades de las superficies y sus uniones; **de volumen**, los objetos son representados en términos de sólidos tales como cilindros generalizados, cubos y bloques.

Cabe mencionar que muchos de estos algoritmos existentes se han desa-

rrollado originalmente para ser usados sobre imágenes (información 2D), sin embargo, ha sido posible extender su uso para extraer descriptores de modelos 3D y aplicar los algoritmos de búsqueda de correspondencias para realizar el reconocimiento.

A partir de lo expuesto anteriormente, a continuación haré un recorrido por algunos de los diferentes enfoques clásicos más relevantes desarrollados para el reconocimiento de objetos y que han marcado pauta para trabajos posteriores, sus ventajas y desventajas, igualmente mencionaré brevemente los avances hechos sobre el uso de HMMs tanto para el caso 2D como 3D, no se pretende profundizar en algoritmos sino dar una visión general del estado del arte.

Nevatia, Binford, Marr y Brooks, entre otros, popularizaron el enfoque de reconocimiento usando partes volumétricas [1]. Este enfoque busca reconocer objetos utilizando cilindros y otras formas bien definidas para lograr un acercamiento a la forma del objeto a describir, la cantidad, tipo y tamaño de estas formas especificarán a dicho objeto para su posterior reconocimiento. La complejidad de las formas reducen significativamente el número necesario para representar al objeto y con ello se hace eficiente el empatado a la hora de establecer correspondencias, no obstante, se dificulta descubrir que partes forman un objeto. Por otro lado, existe el enfoque basado en vistas/apariencia con el cual es posible definir un objeto utilizando características de más bajo nivel, como líneas, bordes, esquinas, curvas, entra otras, pero se genera una enorme cantidad de estas características, dificultando por consecuencia el empatado de las mismas y haciendo necesario el uso de algoritmos de depuración que afinen el resultado del empatado para eliminar posibles falsos positivos. Las características para éste mismo enfoque basado en vistas/apariencia se adquieren no solo de formas básicas como las mencionadas anteriormente, también es posible obtenerlas a partir de valores de pixel en relación con su vecindario para el caso de imágenes.

Un problema observado en el enfoque basado en vistas/apariencia del párrafo anterior es la cantidad de características que se llegan a extraer de una imagen o modelo 3D, pudiendo ser cientos o miles y haciendo extremadamente tardado el cálculo de las correspondencias. Ante este problema surge una técnica llamada Organización Perceptual [1] con la que se intenta modelar la habilidad del sistema de visión humano para detectar propiedades no accidentales de características de bajo nivel y agruparlas según ciertos

factores comunes a fin de construir representaciones de objetos más compactas, de esta forma conseguimos menos características representativas de un objeto y por consecuencia los cálculos para encontrar las correspondencias se reducen significativamente.

Como una variante del enfoque basado en partes volumétricas surgen los modelos deformables [4], la idea en este enfoque es generar modelos 2D o 3D, según sea el caso, que se puedan deformar para ajustarse a los objetos que se intentan especificar, estos modelos se pueden deformar tanto como se permita según los parámetros que lo definen, por ejemplo, una esfera puede deformarse en tamaño y forma, logrando incluso deformarse hasta parecer un elipsoide. Como puede observarse, este enfoque es un paso más adelante del enfoque basado en partes volumétricas y la descripción de los objetos que se desean reconocer se consigue según el número de modelos y el valor de los parámetros de estos que se han variado. El problema con este enfoque es el mismo que se tiene con el de partes volumétricas, en la práctica no es sencillo ajustar las formas complejas.

Una propuesta interesante para el uso de modelos deformables es la definición de los parámetros que deforman estos modelos a partir del Análisis de Componentes Principales (PCA) de diversas capturas del objeto en fase de entrenamiento, de este modo conseguimos un modelo que se deformará solamente dentro de los alcances obtenidos en el entrenamiento de los parámetros. El Análisis de Componentes Principales se puede realizar no solo sobre la forma, sino que también es posible realizarlo en base a la variación en la textura y efectos de iluminación, consiguiendo un modelo deformable mucho más completo [4].

Una propuesta más, es el reconocimiento basado en características locales, este enfoque ha ganado popularidad en la segunda mitad de los 90's principalmente por su robustez en condiciones de desorden y oclusión parcial [1]. La idea principal, como su nombre lo indica, es la extracción de características de una imagen para posteriormente aplicar técnicas que nos ayuden a conocer el objeto más probable del cual fueron extraídas, una de las técnicas más usadas es la búsqueda de correspondencias entre las características de la imagen entrenada y de la que se quiere reconocer. La forma en que se extraen estas características dice mucho de la robustez de las mismas, un buen algoritmo podría extraerlas de modo que sean invariantes a escala, rotación, cambios de iluminación, incluso tolerantes a ruido, un ejemplo

muy popular es el algoritmo SIFT propuesto por Lowe en su trabajo Objet Recognition from Local Scale-Invariant Features de 1999 [5], en este algoritmo se obtienen un vector de características para cada punto de interés en la imagen, se considera el vecindario del punto de interés a diferentes niveles de resolución así como la orientación local de la imagen para conseguir que sea invariante a la rotación y escala. Actualmente existen muchos otros algoritmos que utilizan la técnica de extracción de características locales, entre las más conocidas están: SURF, BRIEF, BRISK, ORB, FREAK.

2.1.2. Reconocimiento con HMMs

Dentro de las diversas técnicas existentes para obtener el objeto más probable del cual ciertas características fueron extraídas se encuentran los modelos ocultos de Markov (HMMs), misma que ha de ser utilizada en el presente trabajo. En [6] se define un HMM como un proceso estocástico doble con un proceso estocástico subyacente que no es observable (está oculto), pero puede ser observado a través de otro conjunto de procesos estocásticos que producen una secuencia de símbolos observables.

Una de las primeras aplicaciones en donde se usaron los Modelos Ocultos de Markov fue en reconocimiento del habla, poco después se comenzaron a utilizar en el análisis de secuencias biológicas, como por ejemplo, secuencias de ADN, también se han usado en el campo de las finanzas y en seguridad de redes para detectar anomalías en el tráfico de red. En el campo de la visión computacional comenzaron a ver su importancia en el reconocimiento del lenguaje de señas, análisis del comportamiento humano con respecto a su movimiento, reconocimiento de poses y cualquier otra aplicación donde una secuencia de observaciones en el tiempo fuera evidente, sin embargo, la propiedad importante de los Modelos Ocultos de Markov como se analizará un poco más adelante, no es la secuencia de tiempo, sino la independencia condicional de cual haya sido la historia del sistema hasta llegar al estado actual y casos en los que se pueda aprovechar esto existen muchos. El reconocimiento facial y de objetos en el área de Visión Computacional ha aprovechado esta propiedad de independencia condicional de los Modelos Ocultos de Markov, utilizando una relación espacial entre variables en lugar de una temporal.

Respecto al uso de los HMMs para reconocimiento de objetos, estos fueron propuestos por primera vez en [7] y extendida posteriormente a objetos

3D en [8][9][10]. En la propuesta [8], Young Kug Ham, Kil Moo Lee y Rae-Hong Park hacen uso de características 3D tales como superficies de área, tipos de superficie y longitudes de línea extraídas de imágenes de rango, este tipo de imágenes poseen la peculiaridad de representar con diferentes niveles de gris cada pixel dependiendo la distancia a la cual se encuentra de un punto en particular. En [10] se hace una proposición interesante del uso de los HMMs para reconocimiento, los autores realizan el entrenamiento de un objeto completo consiguiendo buenos resultados con 8 diferentes tomas del mismo (imágenes RGB), además ellos proponen crear un modelo para cada una de las vistas del objeto y la representación de sus observaciones son coeficientes wavelets de pequeñas sub imágenes extraídas, con estos coeficientes se realiza el entrenamiento para el modelo correspondiente. A pesar de los buenos resultados reportados, esta propuesta soporta su eficiencia en la calidad de la imagen y de la iluminación, condiciones no siempre disponibles cuando se trata de usar la solución en un robot bajo condiciones ambientales inciertas.

Cabe aclara que aun cuando los HMMs aplicados al reconocimiento de objetos 3D ya han sido estudiados, se posee muy poca literatura al respecto y no se tiene conocimiento de que hayan sido implementados realmente en casos prácticos como lo es un robot.

2.1.3. Reconocimiento en la robótica

En la industria como en otras áreas existen procesos que requieren de la capacidad visual, estos a menudo resultan tediosos, monótonos o peligrosos, en un principio estas labores eran tarea exclusiva de humanos, pero gracias al avance en el reconocimiento aplicado a sistemas mecánicos se ha logrado una automatización que no solo ha acelerado la labor, también ha reducido errores como consecuencia del cansancio visual así como daños en la salud de los trabajadores.

La mayoría de las veces cuando los algoritmos de reconocimiento o de visión en general se utilizan para automatizar procesos, estos se usan sobre ambientes en los que es posible manipular las condiciones externas para conseguir los mejores resultados, pero también existen sistemas sobre los cuales no es posible controlar estas condiciones, sistemas autónomos que interactúan con ambientes más complejos como los robots por ejemplo.

Estos sistemas de visión desarrollados para robots han usado los algo-

ritmos de reconocimiento tal como se propusieron en la literatura, pero se ha visto una deficiencia en ellos por aspectos no considerados, propios de los robots autónomos que interactúan con ambientes altamente variantes, ejemplos de estos aspectos son: cambios en la iluminación, oclusión parcial, ruido en la adquisición de los datos. Por estas debilidades se sigue buscado mejorar el reconocimiento de objetos aplicando diferentes técnicas, el uso de datos de profundidad es una de ellas, dispositivos como el Kinect de Microsoft han puesto al alcance de muchas instituciones académicas y científicas información RGB-D que antes solo era posible conseguir con dispositivos difíciles de adquirir. Estos datos de profundidad aportaron la información requerida para una eficiente segmentación de objetos y un conocimiento más preciso sobre su geometría.

Los robots de servicio Judy y Justina desarrollados en el laboratorio de Biorobótica que dirige el Dr. Jesús Savage Carmona, son un ejemplo de la evolución en lo que respecta al sistema de visión que incorpora, los algoritmos implementados se tuvieron que adaptar a las condiciones en las que el robot opera, actualmente los algoritmos de reconocimiento implementados utilizan información de profundidad para realizar la segmentación de los objetos e información RGB para el reconocimiento, a pesar de obtener una buena segmentación, el reconocimiento aun presenta problemas.

2.2. Nube de puntos

Para representar una imagen en la computadora, es posible hacerlo mediante un arreglo bidimensional en donde cada localidad simboliza un pixel de la imagen, pero cuando se trata de manejar datos de tres dimensiones es necesario utilizar alguna otra forma de representación. Tres de las opciones de representación más conocidas son: voxels, mallas y nubes de puntos. De las tres alternativas anteriores, la nube de puntos podría considerarse como la más pura de todas, esta puede ser obtenida directamente desde un hardware especializado sin necesidad de realizar un procesamiento extra complejo.

2.2.1. Definición

Una nube de puntos se puede definir como una colección de vértices sin aparente relación entre ellos y que en su forma más simple normalmente

poseen información de localización en un espacio tridimensional (X, Y y Z), teniendo inicialmente al dispositivo de captura como el punto de referencia. Las nubes de puntos no suelen utilizarse directamente en aplicaciones, estas se procesan para obtener representaciones que aportan más información, como las mallas o superficies. El nombre que se ha dado a esta forma de representación es como consecuencia visual, al mostrar los vértices en un visualizador, la falta de conectividad entre ellos hace que se vean como puntos flotando [11].

En la práctica, a menudo se diferencian entre dos tipos de nubes, organizadas y desorganizadas, la única diferencia es que las primeras, como su nombre lo indica, están organizadas en una estructura bidimensional con un ancho y un alto, similar a una imagen.

2.2.2. Hardware y adquisición

Existen diferentes dispositivos para obtener la nube de puntos correspondiente a un ambiente, los más comunes se apoyan en el uso de luz láser, pero es diferente la manera en que cada uno procesa la información que obtiene. Las técnicas de adquisición dependen de la tecnología que utilizan, de los dispositivos que hacen uso de luz láser las tres técnicas más frecuentes según [12] y [13] son:

- Escaneo por tiempo de vuelo: el método de medición que utiliza es, calcular el tiempo que tarda la luz en ir y regresar desde la fuente láser hasta la superficie y con ellos estimar la distancia, después es posible realizar una triangulación utilizando el ángulo que existe entre una emisión y otra y de este modo obtener los datos XYZ para cada punto en la superficie.

- Escaneo basado en comparación de fase: esta técnica de medición es muy similar a la anterior, la diferencia radica en la manera que calcula la distancia, esta se consigue por el cambio en la señal de la luz láser emitida con respecto a la recibida.

- Escaneo basado en luz: este tipo de escaneo utiliza un proyector de luz infrarroja y una cámara también de luz infrarroja que captura la forma en que la luz proyectada se deforma en la superficie objetivo. La cámara conoce la manera en que la luz se ha emitido y lo compara con

lo observado, diferencias importantes en la comparación representan cambios en la superficie, es decir, áreas lejanas o cercanas. Cuando la luz proyectada es de un patrón conocido, suele llamarse luz estructurada. Los dispositivos de luz estructurada suelen ser más sensibles que los dos anteriores a la luz solar y a materiales altamente reflejantes.

Otra técnica también muy utilizada que no depende del uso de luz láser es la adquisición de nubes de puntos usando cámaras estereoscópicas, incluso es posible combinarla con técnicas de odometría. Este enfoque resulta más natural debido a la similitud que tiene con la forma en que lo hacemos los humanos, sin embargo a menudo resulta en nubes demasiado ruidosas y con muy pocos detalles para objetos pequeños.

A pesar de estas deficiencias, la estereoscopía fue en un principio muy utilizada debido a lo económico que resultaba la solución en relación con los costos en los que se incurría al utilizar escáneres 3D, pero en años recientes (2010 a la fecha de esta tesis), gracias al surgimiento del dispositivo Kinect de la empresa Microsoft, primer cámara RGB-D comercial de bajo costo, se amplió el uso de este tipo de dispositivos y las posibilidades en la investigación científica y desarrollo de aplicaciones en este campo se extendieron.

Las especificaciones técnicas de la cámara Kinect de Microsoft en su primera versión (Modelos 1.5, 1.6, 1.7 y 1.8) son [URL1][14]:

- Técnica: Luz estructurada
- Tamaño de Imagen RGB: 640 x 480 a 30 fps
- Tamaño de mapa de profundidad: 640 x 480 a 30 fps
- Ángulo de visión horizontal: 57 grados
- Ángulo de visión vertical: 43 grados
- Rango de visión de profundidad: 1.8 a 3.5 metros

Para la adquisición de datos desde la cámara Kinect existen drivers, middlewares y SDKs que extraen la información desde la cámara y nos la proporcionan como estructuras para ser fácilmente utilizadas; en estas estructuras se organiza la información RGB y la del mapa de profundidad. Los drivers más conocidos incluyendo sus respectivos middleware y SDK son:

OpenNI (1, 1.5 y 2), Microsoft Kinect SDK y OpenKinect; en la siguiente sección se describirá un poco más de OpenNI, ya que ha sido la opción elegida por tratarse de software open source desarrollado por la empresa que produce directamente el hardware.

2.2.3. Software de procesamiento

OpenNI (v1.5.2)

OpenNI (Open Natural Interaction) v1.5 es un framework Open Source distribuido por la organización del mismo nombre bajo la licencia GNU Lesser General Public License (LGPL), cuyo objetivo es facilitar el desarrollo de aplicaciones utilizando información desde diversos dispositivos de adquisición de datos 3D. Proporciona las estructuras de datos necesarias y las interfaces que deben ser implementadas para generar o usar estas estructuras de datos; incluye interfaces para los componentes de más bajo nivel hasta los de más alto, estos componentes son conocidos como nodos de producción, lo que se traduce en drivers para el soporte de los dispositivos, así como los algoritmos capaces de proporcionar información más compleja, como por ejemplo, detección de gestos o posturas.

La manera en que OpenNI v1.5 está diseñado, permite la creación de nodos de diferentes tipos que interactúan compartiendo información, generando con esto un grafo con un flujo ascendente. Gracias a que cada uno de estos nodos implementa una de las interfaces proporcionadas por OpenNI, se asegura interoperabilidad entre ellos. También existe la posibilidad de utilizar desde la aplicación algún nodo en específico, por ejemplo, aquel que interactúa directamente con los dispositivos y que proporciona la imagen RGB u otro que proporcione el mapa de profundidad.

La imagen (2.1) extraída de [URL2] muestra la arquitectura de OpenNI, los componentes Middleware son los algoritmos que usan los datos obtenidos de los dispositivos y la procesan para entregar información de más alto nivel, tanto las aplicaciones finales de usuario, como los componentes Middleware y los dispositivos, implementan alguna de las interfaces proporcionadas. En conclusión, OpenNI es un framework que provee las herramientas necesarias para establecer una comunicación adecuada entre sensores y aplicaciones de usuario final. En caso de requerir más información, referirse a la guía de usuario de OpenNI [URL2].

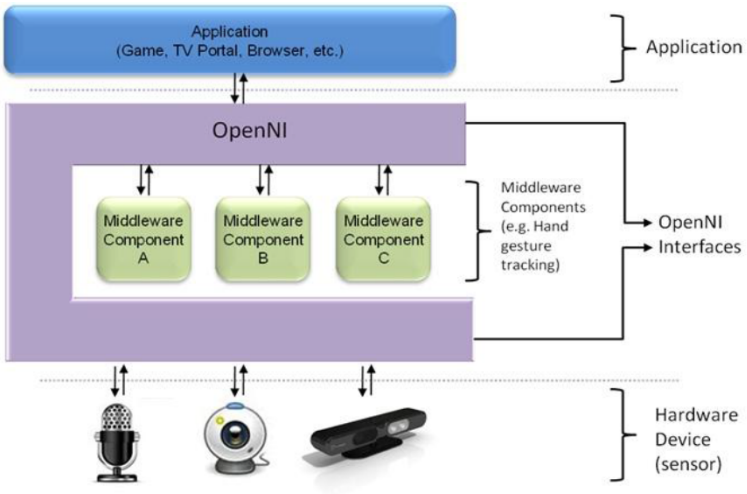


Figura 2.1: Arquitectura OpenNI.

A pesar de que la organización PrimeSense, principal miembro de OpenNI, fue comprada por la empresa Apple, el proyecto fue restructurado y es ahora soportado por Occipital INC bajo el nombre de OpenNI2. Ver [URL3] para más información.

OpenCV

OpenCV (Open Source Computer Visión) es una librería de algoritmos de visión computacional para imágenes, es Open Source y se distribuye bajo la licencia BSD, por lo tanto, es de libre uso académico y comercial. Contiene más de 2500 algoritmos optimizados y varios tipos de datos para el manejo de imágenes. Actualmente existen interfaces para los lenguajes C, C++, Python y Java y puede funcionar bajo Windows, Linux, Android, iOS y Mac OS. [URL4][URL5]

PCL

PCL (Point Cloud Library) es un framework Open Source que contiene diversos algoritmos para el procesamiento de nubes de puntos, es distribuido bajo la licencia BSD, por lo tanto, es de libre uso académico y comercial. Es multiplataforma, por lo que es posible usarlo en Windows, Linux, Android y Mac OS. Dentro de los algoritmos implementados existen de filtrado, de estimación de características, reconstrucción de superficies, registro, ajus-

te de modelo y segmentación; estos algoritmos son programados, revisados y mejorados por la comunidad de investigación en robótica y percepción [18][URL6].

2.2.4. Preprocesamiento

El preprocesamiento es una de los pasos más importantes cuando se trabaja con datos obtenidos a través sensores, particularmente imágenes o nubes de puntos. La información a menudo viene acompañada de ruido generado por el dispositivo de captura o por algunos elementos no deseados en la escena. Aquí es donde se preparan los datos para mejorar y agilizar el cálculo de los resultados del procesamiento final.

Se puede hablar de dos etapas dentro del preprocesamiento: el mejoramiento de los datos y la segmentación.

Mejoramiento de los datos

En esta etapa se elimina el ruido, se minimizan los detalles menos importantes y resaltan aquellos relevantes, para conseguirlo es necesario utilizar filtros principalmente y algunas otras técnicas, la elección de estos dependen del tipo de datos que se analizan, sus características y de la naturaleza del procesamiento que se realizará posteriormente.

En nubes de puntos es común encontrar valores atípicos (outliers) como consecuencia de una mala lectura del sensor y en dispositivos como el Kinect V1 de Microsoft, uno de los problemas es la mala calidad de la nube que proporciona, ya que posee altas variaciones en los puntos de una superficie homogénea. A consecuencia de lo anterior, esta etapa ha resultado de gran relevancia en este trabajo.

A continuación se listarán los filtros y técnicas más relevantes usadas en nubes de puntos y se explicará brevemente el aspecto que mejoran:

- Eliminación condicional de valores atípicos. Con esta técnica se busca eliminar todos aquellos puntos que resultaron de una mala lectura del sensor, es simple y eficiente; consiste en remover de la nube los puntos que no cumplan con una condición, la cual puede ser, que uno de los componentes (X, Y o Z) estén fuera de un rango establecido o que dentro de un radio definido del punto no se posea el número mínimo deseado de vecinos.

- Eliminación de valores atípicos usando técnicas de análisis estadístico. Similar a la anterior, difiere en la forma como se establece la condición; desde cada punto en la nube se calcula la distancia media a su vecindario (el número de vecinos es definido manualmente), se supone una distribución Gaussiana en dicho vecindario y se establece la condición con respecto al número de desviaciones estándar máximas permitidas a partir de la media calculada (el número de desviaciones estándar máximo se define manualmente). La distancia de cada punto del vecindario al punto de consulta se calcula nuevamente y si supera el umbral de las desviaciones estándar permitidas, el punto se elimina.

- Filtro mediana. Suele utilizarse para reducir valores atípicos menos significativos, no los elimina, por lo que es conveniente utilizarlo después de aplicar alguno de los anteriores. Se aplica sobre una componente a la vez (X, Y o Z), sustituyendo el valor de dicha componente en el punto objetivo por el valor de la mediana calculada con los puntos de su vecindario. Este filtro funciona mejor en nube de puntos organizadas.

- Filtro media. La técnica es similar a la del filtro mediana, la diferencia es que el valor de la componente del punto objetivo ahora se sustituye por la media, también calculada con el vecindario. Es ideal para reducir ruido y funciona mejor sobre nube de puntos organizadas.

- Suavizado con algoritmo MLS. Moving Least Squares (MLS) es un algoritmo de reconstrucción que puede ser utilizado para eliminar ruido que no es posible con otras técnicas. MLS aproxima una pequeña superficie a través de una interpolación polinomial de orden alto utilizando puntos de un vecindario, una vez conseguida la representación polinómica de la superficie, elimina los puntos originales y remuestrea obteniendo nuevos puntos libres de ruido, por último repite nuevamente el proceso para conseguir una mejor aproximación. El algoritmo incluso consigue rellenar huecos si los hubiera.

Segmentación

El tema de la segmentación en visión computacional es muy amplio y se han dedicado capítulos enteros para explicar los diferentes algoritmos. En [3] se define la segmentación como: "Proceso por el cual se extraen de la

imagen cierta información subyacente para su posterior uso". Una defición de segmentación más enfocada hacia la nube de puntos se encuentra en [19], como: "La segmentación de una nube de puntos corresponde a la subdivisión de la nube en sub partes que probablemente representan diferentes objetos.".

Segmentar requiere de técnicas que permitan agrupar aquellas unidades (píxeles o vértices en una nube de puntos, por ejemplo) que tengan algo en común y separar las que no, por tanto, se requiere de alguna métrica para cuantizar la similitud. En la literatura, los autores concuerdan en dos enfoques bien definidos, uno basado en regiones (similitud) y otro orientado a bordes (discontinuidad) [3][20][21]. Para cada uno de los enfoques mencionados existen diferentes algoritmos, a continuación se mencionan algunos para procesamiento de imágenes, organizados según su enfoque:

- Basados en regiones (similitud):
  - Umbrales por intensidad
  - Crecimiento de regiones
  - División de regiones
  - Similitud en colores
- Basados en bordes (discontinuidad):
  - Algoritmos de primera y segunda derivada
  - Detector de Canny
  - Ajuste de curvas
  - Contornos activos

La segmentación en nubes de puntos requiere técnicas similares a las utilizadas en imágenes, incluso, para nubes de puntos organizadas (aquellas que se organizan en una matriz n x m), es posible adaptar la mayoría de los algoritmos arriba mencionados. Por otro lado, también existen técnicas basadas en regiones que aprovechan características de las nubes, por ejemplo, clusterización o algoritmos como ICP y RANSAC [19].

Según [19], la segmentación en nubes de puntos puede hacerse con o sin conocimiento previo de lo que se desea segmentar. Algoritmos de clusterización por ejemplo, son capaces de trabajar sin supervisión, dividiendo y agrupando puntos según algún criterio (distancia o similitud) y encontrando

conjuntos que probablemente se tratan de objetos independientes; el algoritmo más famoso de clusterización se llama K-Medias, se trata de un algoritmo iterativo que divide un conjunto de puntos en K bloques, asignando cada punto al bloque con la media más cercana.

En contraste, si se posee conocimiento previo de lo que se desea segmentar, como forma, tamaño o posición estimada, existen algoritmos capaces de ajustar un modelo a un conjunto de datos observados; dos muy conocidos son ICP y RANSAC, en una nube de puntos, estos algoritmos pueden encontrar un modelo en la escena, como planos, curvas, superficies, ya sea parametrizables o no. Ambos algoritmos requieren un método de inicialización, RANSAC propone hacerlo aleatoriamente, también es posible utilizar estos algoritmos juntos para conseguir mejores resultados. Más adelante se detallará el algoritmo RANSAC, utilizado en la segmentación del plano sobre el cual se ubican los objetos a reconocer en este trabajo.

2.2.5. Caracterización

El proceso para caracterizar una imagen o nube de puntos, se reduce a detectar puntos clave o keypoints y a extraer sus características (descripción), los Keypoints son elementos visualmente distintivos y estables, como las esquinas. Posterior a la detección, se extrae información del vecindario de los keypoints para describirlos y poderlos comparar en un futuro proceso de reconocimiento, los buenos algoritmos de descripción crean descriptores robustos a variaciones en la iluminación, cambios en la rotación y escala. Estos puntos descritos o descriptores se representan en forma de vector y el conjunto de ellos caracterizan los datos originales.

Para encontrar keypoints existen diversos algoritmos, uno muy conocido y fácilmente implementable para nubes de puntos es el detector de esquinas de Harris que básicamente busca puntos con cambios significativos en todas las direcciones. PCL incluye algunos otros, por ejemplo: AGAST, BRISK, ISS, NARF, SIFT, SUSAN y Trajkovic [URL7].

Para la descripción también se han desarrollado muchos algoritmos. Como se mencionó anteriormente, estos deben considerar el vecindario de los keypoints y ser robustos. Algunos de los algoritmos de descripción existentes e implementados en PCL se mencionan a continuación [URL8]:

- 3D Shape context.

- 3 Moment invariants.
- Principal surface curvatures.
- BRISK.
- Clustered Viewpoint Feature Histogram (CVFH).
- Fast Point Feature Histogram (FPFH).
- Global Fast Point Feature Histogram (GFPFH).
- Global Radius-based Surface Descriptor (GRSD).
- Point Feature Histogram (PFH).
- Viewpoint Feature Histogram (VFH).
- Normal Aligned Radial Features (NARF).
- Signature of Histograms of OrienTations (SHOT).

2.3. Modelos Ocultos de Markov

Podemos describir los Modelos Ocultos de Markov como autómatas de estados finitos que en cada transición de estado generan un símbolo observable. Las transiciones entre estados se dan de forma probabilística, al igual que la emisión de símbolos en cada estado, hablamos por lo tanto, de un doble proceso estocástico, estos son: la transición entre estados y la generación de símbolos en cada transición; la secuencia de estados no se conoce, es oculta, esta es una razón por la cual este tipo de autómata recibe el nombre de Modelo Oculto de Markov.

Esta sección contiene los elementos necesarios para la comprensión de los modelos ocultos de Markov y se espera que sirva como antesala a la descripción del sistema desarrollado.

2.3.1. Definición Formal

Un Modelo Oculto de Markov(HMM) es un doble proceso estocástico, con un proceso estocástico implícito que no es observable (es oculto), y que puede ser observado a través de otro conjunto de procesos estocásticos que producen la secuencia de observaciones [6][22].

Este modelo consigue su nombre de dos propiedades que lo definen [23]:

1. Asume que la observación en el tiempo t fue generada por algún proceso cuyo estado St está oculto para el desarrollador.

2. Asumen que el estado de este proceso oculto satisface la propiedad de Markov, esta es: Dado el valor de St-1, el estado actual St es independiente de todos los estados antes de t-1.

2.3.2. Elementos de un Modelo Oculto de Markov

Rabiner en [6][22] describe los siguientes elementos de un HMM:

1. Un número N de estados, el conjunto se define como $S = \{s_1, s_2, s_3, ..., s_N\}$ y un estado al tiempo t como $q_t \in Q$.

2. Un conjunto de M observaciones, es decir, los símbolos que produce el modelo en cada transición de estado. En este trabajo se definirán como $V = \{v_1, v_2, v_3, ..., v_M\}$ y una observación al tiempo t como $o_t \in O$.

3. La tabla A de probabilidades de transiciones de estados, esta se describe como $A = \{a_{ij}\}, a_{ij} = P[q_{t+1} = s_j \mid q_t = s_i], 1 \leq i, j \leq N$, donde a_{ij} define la probabilidad de ir al estado s_j en el tiempo $t + 1$ dado que nos encontramos en el estado s_i en el tiempo t. Algunas transiciones entre estados tendrán un valor de 0 dependiendo el tipo de HMM elegido, más adelante se hablará de ellos.

4. La distribución de probabilidad de la observación de símbolos en un estado. Entonces para $B = \{b_j(k)\}$ tenemos que $b_j(k) = P[v_k \ a \ t \mid q_t = s_j], 1 \leq j \leq N, 1 \leq k \leq M$ es la probabilidad de ver el símbolo v_k en el estado s_j al tiempo t.

5. Las probabilidades iniciales de los estados, $\Pi = \{\pi_i\}$ donde $\pi_i = P[q_1 = s_i], 1 \leq i \leq N$ representa la probabilidad de que en el tiempo 1 se encuentre en el estado s_i.

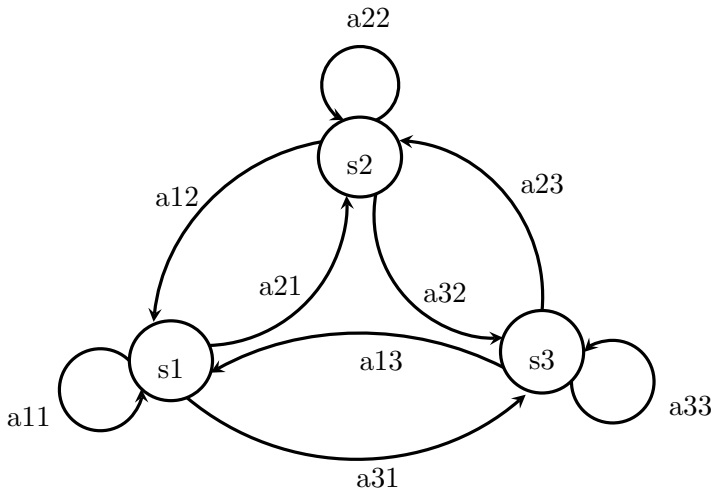
   Entonces, un HMM se puede definir completamente como $\lambda = (\Pi, A, B)$.

2.3.3. Tipos de Modelos

El diseño del modelo a usar depende directamente de las características del problema, en [6][22] se presentan tres diseños que vale la pena mencionar para tener una idea de como estos pueden variar para adaptarse a las necesidades:

- Ergódico: es el tipo más general de HMM, desde cada uno de los nodos (estados) se puede llegar a cualquier otro en un número finito de pasos (estríctamente hablando), sin embargo, el término se suele utilizar para definir a aquellos tipos de HMM en donde todas las transiciones son posibles.

  La imagen 2.2 muestra el diagrama de un HMM ergódico y seguido se anexa su respectiva tabla de transiciones.



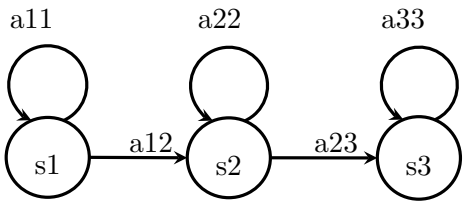
$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Figura 2.2: HMM Ergódico.

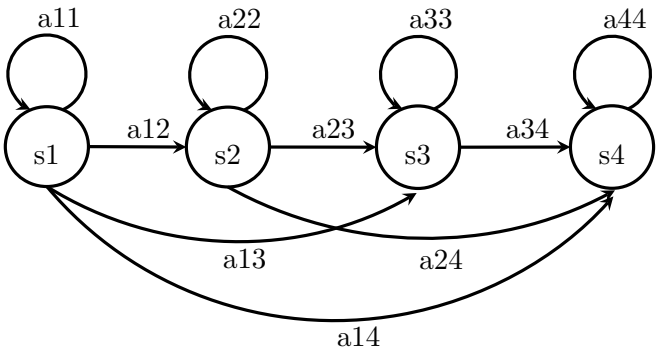
- De izquierda a derecha o de Bakis: son aquellos en donde solo existen transiciones desde un nodo de índice inferior a uno superior, por lo que se debe cumplir que $a_{ij} = 0, i > j$. Este tipo de HMM son muy

utilizados pues se adecuan muy bien a problemas que generan una secuencia de observaciones a través del tiempo, como el habla por ejemplo.

Las imágenes 2.3 y sus respectivas tablas de transiciones muestran dos ejemplos de este tipo de HMM. Como se puede observar, en la tabla de transiciones se forma una matriz triangular superior.



$$A = \begin{bmatrix} a_{11} & a_{12} & 0 \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{bmatrix}$$

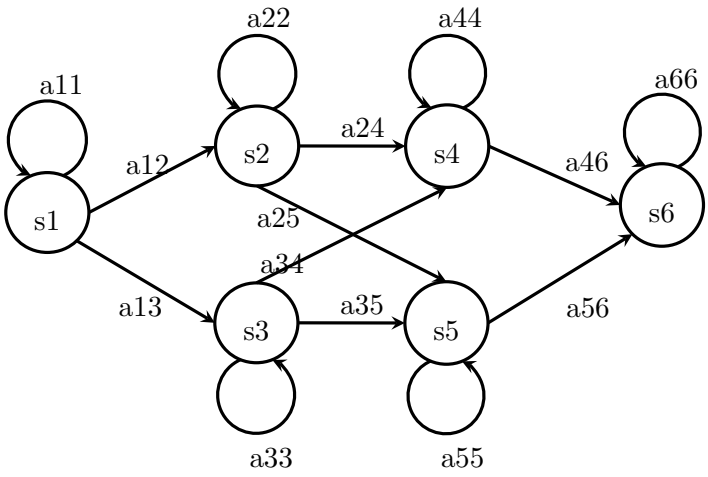


$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}$$

Figura 2.3: HMM Izquierda - Derecha.

- De izquierda a derecha con rutas paralelas: este tipo de modelo es de modo estricto uno de izquierda a derecha, pero puede alternar entre

dos rutas. Se menciona a modo de ejemplo para ilustrar la flexibilidad en el diseño de los modelos.



$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 & 0 \\ 0 & a_{22} & 0 & a_{24} & a_{25} & 0 \\ 0 & 0 & a_{33} & a_{34} & a_{35} & 0 \\ 0 & 0 & 0 & a_{44} & 0 & a_{46} \\ 0 & 0 & 0 & 0 & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & 0 & a_{66} \end{bmatrix}$$

Figura 2.4: HMM Izquierda - Derecha con rutas paralelas.

2.3.4. Primer problema: problema de evaluación

Dada una secuencia de observaciones $O = o_1, o_2, o_3, ..., o_T$ y un modelo $\lambda = (\Pi, A, B)$, calcular la probabilidad de que las observaciones fueron producidas por el modelo. Se suele utilizar cuando en base del conocimiento se tienen más de un modelo y al realizar un conjunto de observaciones se desea saber a cual de los modelos es más probable que pertenezcan [22].

Solución

Para Rabiner en [22], la solución intuitiva podría plantearse considerando todas las posibles secuencias de estados de tamaño T en el modelo y calcular el resultado del planteamiento sumando las probabilidades conjuntas

de la secuencia de observaciones con cada una de las secuencias de estados posibles, dado el modelo.

La definición matemática de lo anterior sería:

Tenemos

$$O = o_1, o_2, o_3, ..., o_T$$

$$Q = q_1, q_2, q_3, ..., q_T$$

La probabilidad de la secuencia de observaciones para una secuencia de estados:

$$P(O/Q, \lambda) = \prod_{t=1}^{T} p\left(o_t \mid q_t, \lambda\right)$$

Dado que asumimos independencia estadística de las observaciones, tenemos:

$$P(O/Q, \lambda) = bq_1(o_1) \cdot bq_2(o_2) \cdot ...bq_T(o_T)$$

La probabilidad de la secuencia de estados, dado el modelo es:

$$P(Q/\lambda) = \Pi_{q_1} \cdot a_{q_1 q_2} \cdot a_{q_2 q_3} \cdot ...a_{q_{T-1} q_T}$$

La probabilidad conjunta de Q y O dado el modelo:

$$P(O, Q/\lambda) = P(O/Q, \lambda)P(Q/\lambda)$$

Sumamos todas las probabilidades conjuntas para obtener la probabilidad de O dado el modelo:

$$P(O/\lambda) = \sum P(O/Q, \lambda)P(Q/\lambda) \; Para\ toda\ Q\ posible.$$

El algoritmo anterior es poco inteligente y el total de cálculos es del orden de $2T \cdot N^T$.

Existen otros algoritmos basados en la técnica de programación dinámica, llamados forward o procedimiento hacia adelante y backward o procedimiento hacia atrás, los algoritmos que a continuación se presentan fueron extraídos de [22]:

Forward Procedimiento hacia adelante:

Considere la variable $\alpha_t(i)$ como:

$$\alpha_t(i) = P(o_1 o_2 ... o_t, q_t = s_i \mid \lambda) \tag{2.1}$$

la probabilidad de la secuencia parcial de observaciones al tiempo t y encontrarse en el estado s_i, dado el modelo.

La variable anterior se puede resolver inductívamente:

1. Inicialización:

$$\alpha_1(i) = \pi_i b_i(o_1), 1 \leq i \leq N \tag{2.2}$$

2. Inducción:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(o_{t+1}), \tag{2.3}$$
$$1 \leq t \leq T-1$$
$$1 \leq j \leq N$$

3. Conclusión:

$$P(O/\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{2.4}$$

Este procedimiento, como su nombre lo indica, avanza hacia adelante en el tiempo desde t = 1 hasta T, calculando en cada instante de tiempo t y para cada estado i, donde $1 \leq i \leq N$, la probabilidad de la secuencia parcial de observaciones hasta t y que la observación a este tiempo se genere por el estado s_i. Finalmente, las probabilidades de la secuencia de observaciones completa, siendo la observación o_T generada por los diferentes N estados, se suman para obtener $P(O/\lambda)$.

Los cálculos necesarios para este algoritmo están en el orden de N^2T, lo que resulta más eficiente que $2TN^T$ del procedimiento anterior, la clave en

la velocidad es el aprovechamiento de los cálculos anteriores para generar los nuevos.

Con el objetivo de comprender mejor el algoritmo, se ilustrarán los cálculos necesarios para un ejemplo de 3 estados y 3 símbolos observables, se utiliza una tabla como apoyo.

Dada la secuencia de observaciones $O = o_1o_2o_3o_4o_5$, se tiene N = 3 y T = 5. Se asume que se tiene el modelo $\lambda = (\Pi, A, B)$.

	Estados		
	s_1	s_2	s_3
o_1	$\alpha_1(1)$	$\alpha_1(2)$	$\alpha_1(3)$
o_2	$\alpha_2(1)$	$\alpha_2(2)$	$\alpha_2(3)$
o_3	$\alpha_3(1)$	$\alpha_3(2)$	$\alpha_3(3)$
o_4	$\alpha_4(1)$	$\alpha_4(2)$	$\alpha_4(3)$
o_5	$\alpha_5(1)$	$\alpha_5(2)$	$\alpha_5(3)$

El primer renglón, para la observación en el tiempo t = 1 es la inicialización, los siguientes renglones se calculan utilizando los resultados del renglón anterior, para calcular $P(O/\lambda)$ se suman todos los valores de la última fila, es decir, cuando la secuencia de observaciones está completa.

Backward Procedimiento hacia atrás:

similar a forward, la diferencia está en la dirección de los cálculos, mientras forward lo hace desde t = 1 hasta T, backward lo hace de T hasta 1, es decir, comienza por el final. Se presenta debido a que será utilizado en la solución del tercer problema.

Se define la variable

$$\beta_t(i) = P(o_{t+1}o_{t+2}\ldots o_T \mid q_t = s_i, \lambda) \tag{2.5}$$

como la secuencia parcial de observaciones del tiempo t+1 al final, dado el modelo λ y que el estado al tiempo t es s_i.

Resolviendo inductívamente:

1. Inicialización:

$$\beta_T(i) = 1, 1 \leq i \leq N \tag{2.6}$$

2. Inducción:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j), \tag{2.7}$$

$$t = T-1, T-2, ..., 1$$
$$1 \leq i \leq N$$

3. Conclusión:

$$P(O \mid \lambda) = \sum_{i=1}^{N} \beta_1(i) \pi_i b_i(o_1) \tag{2.8}$$

2.3.5. Segundo problema: problema de la secuencia óptima

Dada una secuencia de observaciones $O = o_1 o_2 o_3 o_4 \ldots o_T$ y un modelo $\lambda = (\Pi, A, B)$, encontrar la secuencia de estados $Q = q_1 q_2 q_3 q_4 \ldots q_T$ que mejor explica la secuencia de observaciones.

Encontrar la secuencia óptima de estados dada una secuencia de observaciones y el modelo, suele utilizarse para conocer mejor la estructura de un HMM, también se usa para afinar el entrenamiento a través de un análisis estadístico de los estados.

Solución

El algoritmo de Viterbi, utilizando la técnica de programación dinámica da solución a este problema.

Tenemos un conjunto de observaciones $O = o_1 o_2 o_3 \ldots o_T$, un modelo λ y la incógnita $Q = q_1 q_2 q_3 \ldots q_T$, siendo esta última la secuencia óptima de estados correspondiente al conjunto de observaciones.

Definimos $\delta_t(i)$

$$\delta_t(i) = \max_{q_1,q_2,\ldots,q_{t-1}} P[q_1 q_2 \ldots q_t = i, o_1 o_2 \ldots o_t \mid \lambda] \tag{2.9}$$

como la secuencia de estados de más alta probabilidad al tiempo t, siendo s_i el estado actual.

Por inducción tenemos:

$$\delta_{t+1}(j) = \max_i \left[\delta_t(i) a_{ij}\right] .b_j(o_{t+1}) \tag{2.10}$$

El algoritmo completo extraído de [22] es:

1. Inicialización:

$$\begin{aligned} &\delta_1(i) = \pi_i b_i(o_1), && 1 \leq i \leq N \\ &\psi_1(i) = 0 \end{aligned} \tag{2.11}$$

2. Recursión:

$$\begin{aligned} \delta_t(j) = \max_{1\leq i\leq N} \left[\delta_{t-1}(i) a_{ij}\right] .b_i(o_t), \qquad & 2 \leq t \leq T \\ & 1 \leq j \leq N \end{aligned} \tag{2.12}$$

$$\begin{aligned} \psi_t(j) = \arg \max_{1\leq i\leq N} \left[\delta_{t-1}(i) a_{ij}\right], \qquad & 2 \leq t \leq T \\ & 1 \leq j \leq N \end{aligned} \tag{2.13}$$

3. Conclusión:

$$p^* = \max_{1\leq i\leq N} \left[\delta_T(i)\right] \tag{2.14}$$

$$q_T^* = \arg \max_{1 \leq i \leq N} [\delta_T(i)] \tag{2.15}$$

4. Extracción de la secuencia de estados:

$$q_T^* = \psi_{t+1}(q_{t+1}^*), \qquad t = T-1, T-2, \ldots, 1 \tag{2.16}$$

Como se puede observar, el algoritmo va calculando a cada tiempo t la ruta parcial de estados más probable, cuando o_t es generada por cada uno de los estados. También se conserva el índice del estado de donde se generó la transición para conseguir esa probabilidad mayor. Al final se extrae la secuencia con ayuda de los índices almacenados durante el proceso. Similar al algoritmo forward y backward, los cálculos se pueden organizar como en una tabla de T x N, para almacenar los índices de los estados se requiere una extra de igual tamaño.

2.3.6. Tercer problema: problema del entrenamiento

El ajuste de parámetros es un problema de gran importancia en lo que respecta a HMMs, se refiere al aprendizaje y no existe un procedimiento analítico para resolverlo. Dada una secuencia de observaciones, se deben ajustar los parámetros del modelo $\lambda = (\Pi, A, B)$ tal que maximicen $P(O \mid \lambda)$.

Solución

Baum y sus colegas propusieron un algoritmo iterativo que en cada iteración reestima los parámetros hasta encontrar un máximo local. Rabiner en su trabajo [22] lo describe del siguiente modo:

Algoritmo de Baum-Welch

Se define la variable $\varepsilon_t(i, j)$

$$\varepsilon_t(i,j) = P(q_t = s_i, q_{t+1} = s_j \mid O, \lambda) \tag{2.17}$$

como la probabilidad de encontrarse en el estado s_i en el tiempo t y en el estado s_j en el tiempo t+1.

Describiendo $\varepsilon_t(i,j)$ en términos de las variables forward (2.1) y backward (2.5) estudiadas en el problema de la evaluación:

$$\begin{aligned} \varepsilon_t(i,j) &= \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{P(O \mid \lambda)} \\ &= \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)} \end{aligned} \tag{2.18}$$

donde el numerador es $P(q_t = s_i, q_{t+1} = s_j, O \mid \lambda)$

Definimos la variable $\gamma_t(i)$

$$\gamma_t(i) = P(q_t = s_i \mid O, \lambda) \tag{2.19}$$

como la probabilidad de encontrarse en el estado s_i al tiempo t dado el modelo y una secuencia de observaciones. En términos de las variables forward (2.1) y backward (2.5) queda:

$$\begin{aligned} \gamma_t(i) &= \frac{\alpha_t(i)\beta_t(i)}{P(O \mid \lambda)} \\ &= \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)} \end{aligned} \tag{2.20}$$

Es posible relacionar las variables $\varepsilon_t(i,j)$ y $\gamma_t(i)$ del siguiente modo:

$$\gamma_t(i) = \sum_{j=1}^{N} \varepsilon_t(i,j) \tag{2.21}$$

Sumando $\gamma_t(i)$ desde t=1 hasta T-1 obtenemos una cantidad que representa el número esperado de veces que una transición se hace desde el estado

s_i.

Sumando $\varepsilon_t(i,j)$ desde t=1 hasta T-1 obtenemos una cantidad que representa el número esperado de transiciones del estado s_i al estado s_j.

En resumen:

$$\sum_{t=1}^{T-1} \gamma_t(i) \quad = \text{Número esperado de transiciones de } s_i. \tag{2.22}$$

$$\sum_{t=1}^{T-1} \varepsilon_t(i,j) \quad = \text{Número esperado de transiciones de } s_i \text{ a } s_j. \tag{2.23}$$

La información anterior es necesaria para definir el siguiente procedimiento que reestima los parámetros del modelo:

Sea $\lambda = (\Pi, A, B)$ un modelo inicial propuesto y $\overline{\lambda} = (\overline{\Pi}, \overline{A}, \overline{B})$ el modelo reestimando.

$$\begin{aligned} \overline{\pi_i} \quad &= \text{Número esperado de transiciones del estado } s_i \text{ cuando t = 1.} \\ &= \gamma_1(i) \end{aligned} \tag{2.24}$$

$$\begin{aligned} \overline{a_{ij}} \quad &= \frac{\text{Número esperado de transiciones de } s_i \text{ a } s_j.}{\text{Número esperado de transiciones del estado } s_i.} \\ &= \frac{\sum_{t=1}^{T-1} \varepsilon_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \end{aligned} \tag{2.25}$$

$$\begin{aligned} \overline{b_j}(k) \quad &= \frac{\text{Número esperado de veces en el estado j y observado el símbolo } v_k.}{\text{Número esperado de veces en el estado j.}} \\ &= \frac{\sum_{t=1:o_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \end{aligned} \tag{2.26}$$

En cada iteración del procedimiento anterior se encuentra un nuevo modelo $\overline{\lambda} = (\overline{\Pi}, \overline{A}, \overline{B})$ que según demostraron Baum y sus colegas, puede derivar en alguno de los casos siguientes:

- $\overline{\lambda} = \lambda$, lo que significa que λ define un punto crítico en la función de

probabilidad.

- $P(O \mid \overline{\lambda}) > P(O \mid \lambda)$, dada la secuencia de observaciones O, con el nuevo modelo $\overline{\lambda}$ se consigue más probabilidad que con el anterior λ. Por lo tanto, $\overline{\lambda}$ pasa a ser el nuevo λ y el procedimiento se repite con los nuevos parámetros.

El algoritmo deja de iterar cuando se cumple el primer caso, es decir, existe nula o muy poca diferencia entre $P(O \mid \overline{\lambda})$ y $P(O \mid \lambda)$.

El procedimiento anterior funciona para entrenar un modelo con una secuencia de observaciones, pero en la práctica, uno de los problemas que se tienen al utilizar un modelo de izquierda a derecha o Bakis, es que debido a las restricciones existentes entre las transiciones de sus estados, no es posible regresar cuando ya se ha pasado a un estado futuro, esto provoca que se tengan muy pocas observaciones por estado, entonces para realizar un entrenamiento que aproxime adecuadamente un modelo, es necesario realizar el entrenamiento utilizando múltiples secuencias de observaciones. Para conseguir lo anterior solo será necesario reestimar en cada iteración utilizando todas las secuencias de observaciones que se tengan, una tras otra, las ecuaciones de reestimación entonces se reescriben del siguiente modo (R es el número de secuencias a entrenar y r se usa para referirse a una secuencia en particular):

$$\overline{\pi_i} = \sum_{r=1}^{R} \gamma_1^r(i) \tag{2.27}$$

$$\overline{a_{ij}} = \frac{\sum_{r=1}^{R} \sum_{t=1}^{T_r-1} \varepsilon_t^r(i,j)}{\sum_{r=1}^{R} \sum_{t=1}^{T_r-1} \gamma_t^r(i)} \tag{2.28}$$

$$\overline{b_j}(k) = \frac{\sum_{r=1}^{R} \sum_{t=1:o_t=v_k}^{T_r} \gamma_t^r(j)}{\sum_{r=1}^{R} \sum_{t=1}^{T_r} \gamma_t^r(j)} \tag{2.29}$$

Capítulo 3

DESARROLLO

En este capítulo se mostrarán los detalles de desarrollo del sistema como una propuesta para reconocimiento de objetos basado en la forma, utilizando HMMs discretos y nubes de puntos. Se plantea una arquitectura y se explican sus componentes, finalmente se detallan los comandos para su funcionamiento.

3.1. Hardware utilizado

El hardware usando durante el desarrollo del sistema, así como para las pruebas fue:

- Laptop Alienware marca DELL
  - Procesador Intel Core i7-4700MQ a 2.4 GHz.
  - 8 GB de memoria RAM.
  - Windows 10 64 bits.
  - Disco duro 700 GB.
  - Tarjeta gráfica NVIDIA GForce GTX 765M 2GB GDDR5.
- Dispositivo Kinect V1 modelo 1473(1.5). (Características detalladas en sección 2.2.2.)

3.2. Software utilizado

En esta sección se lista el software utilizado para el desarrollo:

- Sistema Operativo Windows 10 64 bits.

- Microsoft Visual Studio 2010 SP1 (Visual C++ 2010).

- Módulo de Visión: software desarrollado en el laboratorio de biorobótica para su uso en robots, proporciona una interfaz a las tareas que utilizan la información del Kinect y algoritmos de visión computacional, ejecuta las tareas que tiene programadas en forma de solicitudes externas y devolviendo una respuesta. Entre los comandos (tareas) que actualmente posee están: reconocimiento de objetos, de espacios libres, de rostros, detección de bordes de mesas, entre otros.

- OpenNI 1.5.2, OpenCV 2.4.9 y PCL 1.6.0. (Descritos en la sección 2.2.3)

- Estructuras de datos para HMMs desarrolladas por Dekang Lin [URL9].

3.3. Arquitectura del sistema

El diagrama de bloques que se muestra en la imagen 3.1 representa la estructura del sistema propuesto. Como se puede observar, es posible identificar visualmente tres secciones, la primera constituye todas las acciones necesarias para obtener las observaciones de un objeto en la escena, desde conseguir la nube de puntos, hasta describirla; la segunda sección se ubica a la izquierda, define el entrenamiento del modelo utilizando para ello tres secuencias de observaciones del objeto a entrenar y la tercera ubicada a la derecha representa el área de reconocimiento para un objeto observado, dados diversos modelos anteriormente entrenados. Los detalles de cada bloque se describirán a lo largo de este capítulo.

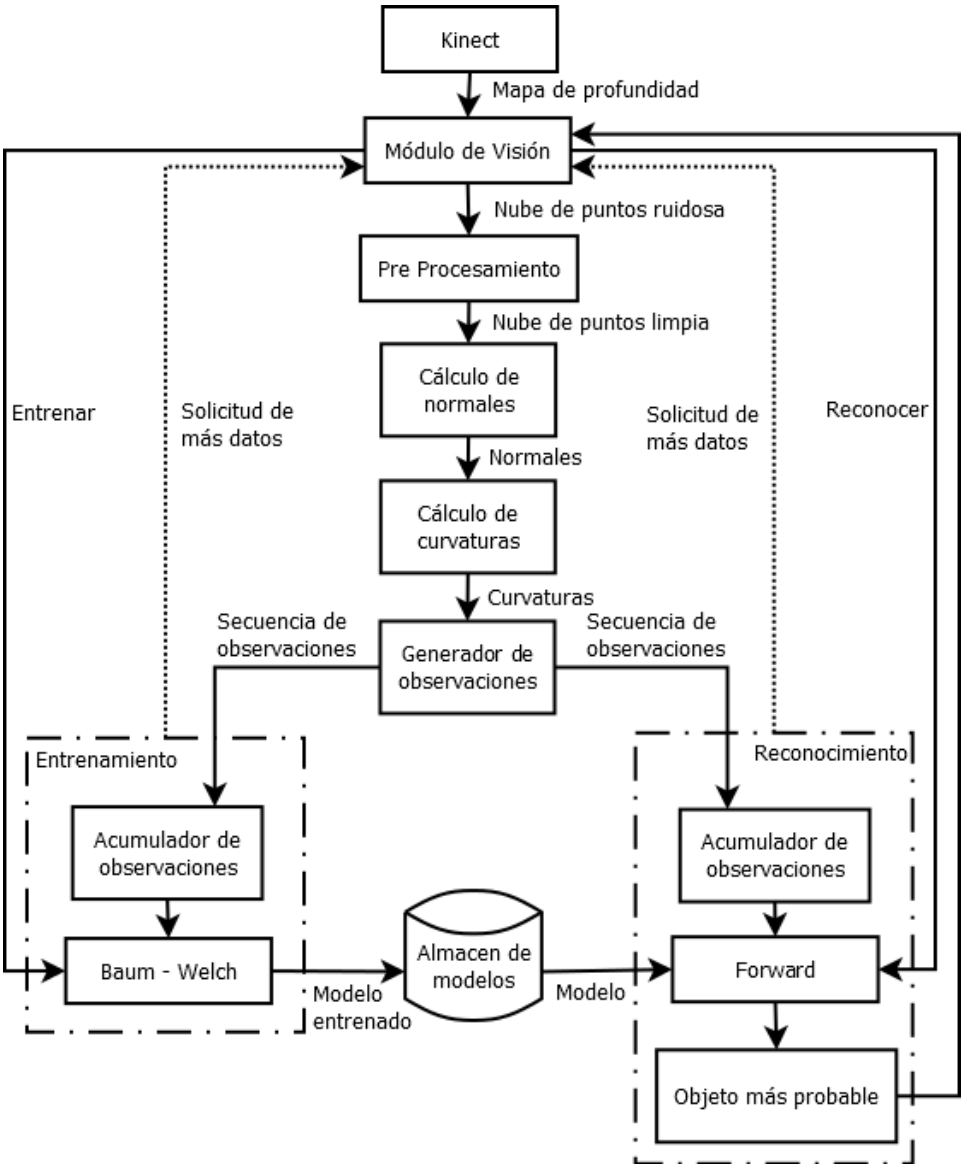


Figura 3.1: Arquitectura del Sistema.

3.4. Adquisición de nube de puntos

El proceso de adquisición de datos se realiza a través del módulo de visión, mismo que a su vez utiliza OpenNI para conectarse con el dispositivo Kinect. El módulo de visión proporciona dos estructuras de datos bidimensionales de tamaño NxM, donde N = 640 y M = 480, una de las estructuras está constituida por elementos que representan puntos 3D, en la otra repre-

sentan pixeles. La estructura de puntos 3D es el mapa de profundidad que corresponde con la de los pixeles, cada punto 3D tienen las componentes X, Y y Z de un espacio tridimensional con origen de coordenadas en el punto de captura, tal como se muestra en la imagen 3.2.

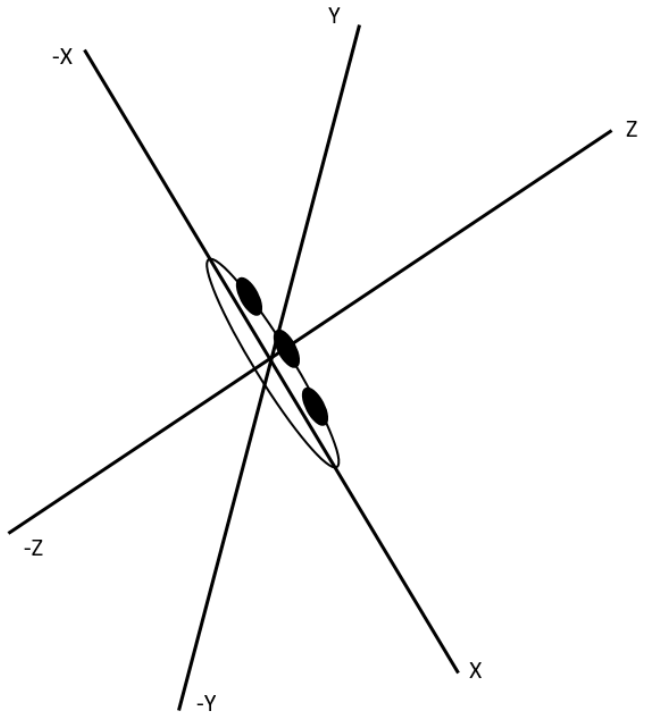


Figura 3.2: Sistema de coordenadas tridimensionales con origen de coordenadas en el Kinect.

Algunos puntos del mapa de profundidad tendrán la componente Z = 0, esto significa que el dispositivo de captura no obtuvo información. Para construir una nube de puntos adecuada, será necesario descartar estos puntos con Z = 0 e incluir los restantes.

3.5. Preprocesamiento de nube de puntos

En esta sección se explican las técnicas utilizadas para la preparación de los datos antes de extraer las características. Se detallan procedimientos de segmentación y mejoramiento de los datos para eliminar ruido. El orden de las etapas en el pre procesamiento es importante para la velocidad del

sistema de reconocimiento en general. Tanto en el entrenamiento, como en el reconocimiento, se ha decidido realizar la segmentación al principio para reducir significativamente la cantidad de puntos en etapas posteriores que a menudo suele consumir más tiempo, como lo es el mejoramiento de los datos.

3.5.1. Segmentación

Para segmentar los objetos de interés en la escena, se aprovechan las características de la misma. A priori conocemos que los objetos estarán sobre un área plana (mesa, piso, alacena, ...), en el tema de segmentación del apartado 2.2.4 se habló sobre técnicas de segmentación que aprovechan conocimiento previo de la nube, entonces se utilizará el algoritmo RANSAC para encontrar el plano y segmentar lo que se encuentra sobre éste.

RANSAC

RANSAC es un algoritmo iterativo que ajusta un modelo parametrizable a un conjunto de datos [25], en cada iteración toma aleatoriamente los datos necesarios para definir el modelo y utiliza alguna medida de error para saber que tanto se ajusta a los datos completos. Ya que funciona de manera aleatoria, no es determinista y por lo tanto no se puede garantizar el mejor resultado, sin embargo, es capaz de mejorar con cada iteración.

El plano es un modelo que se puede definir con tres puntos, en [26] se construye su ecuación general de la siguiente manera:

Dados 3 puntos:

$$O = (X_O, Y_O, Z_O)$$

$$P = (X_P, Y_P, Z_P)$$

$$Q = (X_Q, Y_Q, Z_Q)$$

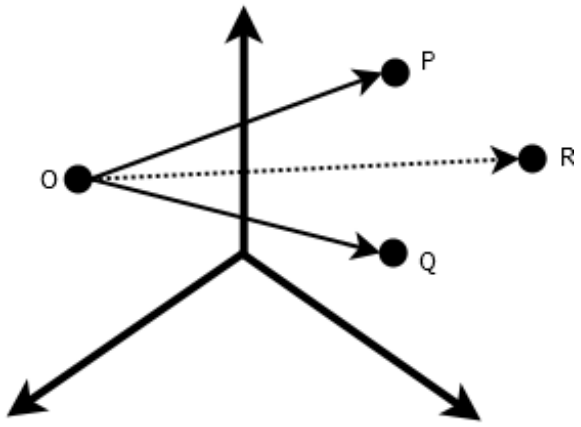


Figura 3.3: Puntos O,P y Q que definen un plano y R que pertenece.

Definimos R como un punto cualquiera que pertenece al plano con componentes (X, Y, Z). Definimos también los vectores directores $\overline{OP}$ y $\overline{OQ}$:

$$\begin{aligned} \overline{OP} &= (X_P - X_O, Y_P - Y_O, Z_P - Z_O) \\ \overline{OQ} &= (X_Q - X_O, Y_Q - Y_O, Z_Q - Z_O) \end{aligned} \tag{3.1}$$

Si el punto R realmente pertenece al plano definido por O, P y Q, entonces el vector $\overline{OR}$ debe ser el resultado de la combinación de los vectores directores, así tenemos:

$$\begin{aligned} &\overline{OR} = \lambda\overline{OP} + \mu\overline{OQ} \\ &(X_R - X_O, Y_R - Y_O, Z_R - Z_O) = \lambda(X_{OP}, Y_{OP}, Z_{OP}) + \mu(X_{OQ}, Y_{OQ}, Z_{OQ}) \end{aligned} \tag{3.2}$$

A partir de esta igualdad es posible obtener el siguiente sistema de ecuaciones:

$$\begin{aligned} X_R - X_O &= \lambda X_{OP} + \mu X_{OQ} \\ Y_R - Y_O &= \lambda Y_{OP} + \mu Y_{OQ} \\ Z_R - Z_O &= \lambda Z_{OP} + \mu Z_{OQ} \end{aligned} \tag{3.3}$$

Para cada punto que pertenezca al plano el sistema tendrá solución. Los

valores de λ y μ son constantes en las tres ecuaciones, ya que el sistema tiene múltiples soluciones, es decir, es singular, el determinante de la matriz ampliada deberá ser igual a cero.

$$det \begin{bmatrix} X_R - X_O & X_{OP} & X_{OQ} \\ Y_R - Y_O & Y_{OP} & Y_{OQ} \\ Z_R - Z_O & Z_{OP} & Z_{OQ} \end{bmatrix} = 0 \tag{3.4}$$

Resolviendo por adjuntos sobre la primer columna:

$$\begin{bmatrix} Y_{OP} & Y_{OQ} \\ Z_{OP} & Z_{OQ} \end{bmatrix} (X_R - X_O) - \begin{bmatrix} X_{OP} & X_{OQ} \\ Z_{OP} & Z_{OQ} \end{bmatrix} (Y_R - Y_O) + \begin{bmatrix} X_{OP} & X_{OQ} \\ Y_{OP} & Y_{OQ} \end{bmatrix} (Z_R - Z_O) = 0$$

Igualamos:

$$\begin{aligned} A &= \begin{bmatrix} Y_{OP} & Y_{OQ} \\ Z_{OP} & Z_{OQ} \end{bmatrix} \\ B &= \begin{bmatrix} X_{OP} & X_{OQ} \\ Z_{OP} & Z_{OQ} \end{bmatrix} \\ C &= \begin{bmatrix} X_{OP} & X_{OQ} \\ Y_{OP} & Y_{OQ} \end{bmatrix} \end{aligned}$$

Sustituyendo y despejando:

$$A(X_R - X_O) - B(Y_R - Y_O) + C(Z_R - Z_O) = 0$$

$$AX_R - AX_O - BY_R + BY_O + CZ_R - CZ_O = 0$$

Hacemos $D = -AX_O + BY_O - CZ_O$ para dejar todo en función del punto desconocido R, quedando la ecuación general del plano como:

$$AX_R - BY_R + CZ_R + D = 0 \tag{3.5}$$

donde:

$$\begin{aligned} A &= Y_{OP}Z_{OQ} - Y_{OQ}Z_{OP} \\ B &= X_{OP}Z_{OQ} - X_{OQ}Z_{OP} \\ C &= X_{OP}Y_{OQ} - X_{OQ}Y_{OP} \\ D &= -AX_O + BY_O - CZ_O \end{aligned} \tag{3.6}$$

La distancia de la perpendicular trazada de un punto al plano es:

$$d = \frac{\mid AX_R - BY_R + CZ_R + D \mid}{\sqrt{A^2 + B^2 + C^2}} \tag{3.7}$$

Con el modelo del plano definido, ahora se muestra RANSAC tal como se implementó en el algoritmo 1.

El algoritmo 1 busca en 100 iteraciones el modelo de un plano que ajuste con la mayor cantidad de puntos posibles (inliers), este será el plano más grande en la escena. Los parámetros iniciales fueron estimados experimentalmente considerando las características propias del ambiente.

Una vez obtenidos los parámetros del modelo, se calculará nuevamente con la ecuación 3.7 la distancia de cada punto en la nube al plano anteriormente estimado, todos los puntos que se encuentren a una distancia menor o igual que 10 mm serán descartados (puntos del plano), el resto se considerarán parte del objeto de interés y se conservarán para la siguiente etapa del proceso.

3.5.2. Mejoramiento de la nube de puntos

Con la nube de puntos segmentada, ahora se procede a realizar el mejoramiento de estos datos con el objetivo de eliminar el ruido y todos los valores que puedan alterar la estimación de los descriptores.

En 3.4 se describió que el módulo de visión entrega la nube en una estructura bidimensional de tamaño NxM, y tal como se menciona en 2.2, este tipo de nube en la literatura suele llamarse nube de puntos organizada. De ahora hasta que se mencione lo contrario se trabajará con esta estructura.

Filtro mediana

Definido en el apartado 2.2.4, este filtro reduce valores atípicos. El filtro mediana se aplicó sobre la componente Z de los puntos (profundidad) uti-

Algorithm 1 RANSAC para planos

Require: Nube de puntos: NubePuntos
Ensure: Constantes del plano: PlanoABCD

```
NUMERO_ITERACIONES = 100
UMBRAL_ES_INLIER = 10mm
INLIERS_CONSEGUIDOS = 0
UMBRAL_MIN_INLIERS = NubePuntos.Tamaño() * 0.1
Inliers = NubePuntos
for it = 0 hasta NUMERO_ITERACIONES do
    TresPuntos = TresPuntosAleatorios(Inliers)
    ABCDTemp = CalcularConstantesPlano(TresPuntos) {Ecuaciones 3.6}
    Inliers.Limpiar()
    for all Punto en NubePuntos do
        DistanciaPlano = Distancia(Punto, ABCDTemp) {Ecuacin 3.7}
        if DistanciaPlano ≤ UMBRAL_ES_INLIER then
            Inliers.Agregar(Punto)
        end if
    end for
    if Inliers.Tamaño() > INLIERS_CONSEGUIDOS then
        INLIERS_CONSEGUIDOS = Inliers.Tamaño()
        PlanoABCD = ABCDTemp
    end if
    if Inliers.Tamaño() < UMBRAL_MIN_INLIERS then
        Inliers = NubePuntos
    end if
end for
return PlanoABCD
```

lizando un kernel de 5x5. Sea f(x, y) la función que define un elemento del mapa de profundidad, dados los componentes (x, y) devuelve un punto 3D en esta posición de la estructura.

Sea:

$$V(x,y) = \{f(x-i, y-j)\}, \quad -2 \leq i,j \leq 2 \tag{3.8}$$

la función que define un bloque de 5x5, formado por el punto (x, y) y su vecindario en el mapa de profundidad.

Tenemos entonces:

$$\begin{aligned} g(x,y) =& mediana\{V(x,y)\} \\ & 2 \leq x \leq N-3 \\ & 2 \leq y \leq M-3 \end{aligned} \tag{3.9}$$

siendo N = 640, M = 480 y g el nuevo mapa de profundidad con el filtro aplicado.

Filtro media o promediador

Definido en el apartado 2.2.4, es un filtro suavizador aplicado después del filtro mediana. Se utilizó un kernel de 7x7 sobre los puntos del mapa de profundidad.

Sea f(x, y) la función para el mapa de profundidad después del filtro mediana y g(x, y) para el mapa después del filtro promediador. Se define matemáticamente como:

$$\begin{aligned} g(x,y) =& \frac{\sum_{j=-3}^{3}\sum_{i=-3}^{3} f(x+i,y+j)}{7x7} \\ & 3 \leq x \leq N-4 \\ & 3 \leq y \leq M-4 \end{aligned} \tag{3.10}$$

Eliminación condicional de valores atípicos por bloque

Esta técnica es similar a la definida en 2.2.4 pero la condición para eliminar valores atípicos trabaja sobre un bloque de 3x3 en el mapa de profundidad y se aplica solo sobre la componente Z de los puntos.

El procedimiento es obtener para cada punto del mapa de profundidad aquellos puntos inmediatamente adyacentes diferentes de cero y calcular hacia cada uno de ellos la diferencia absoluta en la componente Z. Si para todos estos puntos adyacentes se cumple que la diferencia absoluta es mayor que un umbral de 30 mm, entonces el punto central consultado se considera un valor atípico y se descarta.

Con este proceso limpiaremos la nube de algunas anomalías generadas por una mala lectura del sensor. En la imagen 3.4 se ilustra la relación que

define un bloque.

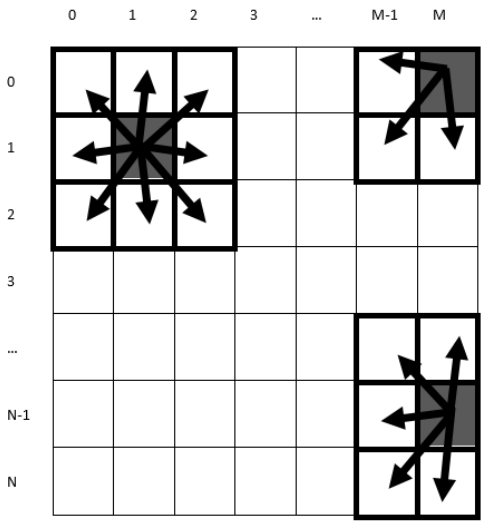


Figura 3.4: Descripción gráfica de un bloque para la eliminación condicional de valores atípicos.

Eliminación de ruido con MLS

A partir de aquí la nube de puntos se migrará a una nueva estructura tipo lista de PCL, ahora se considerará como una nube de puntos desorganizada, la razón principal es que se utilizará un algoritmo implementado en PCL que necesita que la nube tenga esta estructura.

En esta sección se concluirá el pre procesamiento de la nube de puntos con el algoritmo Moving Least Squares (MLS). MLS es un algoritmo de reconstrucción que aproxima una función polinomial de grado alto a una superficie local y remuestrea los puntos, utilizando para ello la función aproximada que ha sido estimada a través del procedimiento de mínimos cuadrados ponderado [28][29].

Se explicará de manera general el algoritmo, pero antes se hace necesario hablar sobre los árboles K dimensionales ó K-d Tree y como pueden ayudar en la búsqueda de los vecinos de un punto dentro de un radio r.

K-d Tree son árboles binarios, con k como la dimensionalidad del espacio de búsqueda, fueron propuestos por Jon Louis Bentley en [27]. Con este tipo

de árboles es posible representar un conjunto de datos k-dimensionales en un árbol. La forma en que se organizan los datos es muy similar a como se realiza con los árboles de búsqueda binaria, la diferencia radica en la llave utilizada en cada nodo, ya que estos nodos representan valores con más dimensiones se usa un dimensión diferente como llave alternándose en cada nivel del árbol, por ejemplo, para organizar puntos de un espacio 2 dimensional, en el primer nivel del árbol utilizamos la componente X para organizar los datos en un sub arbol izquierdo (valores en X menores que en la raiz) y derecho (valores en X mayores que en la raiz), en el segundo nivel del árbol organizamos los datos ahora con respecto al parámetro Y y así alternándose por cada nivel. Para comprender mejor la técnica, en la imagen 3.5 se muestra un pequeño ejemplo.

Cuando se tiene conocimiento del conjunto de datos a organizar, para conseguir un árbol más balanceado suele utilizarse la mediana del grupo de datos que se dividirá como raíz del nodo padre, así para el primer nivel se usa como raíz la mediana de las X's del total de puntos y en el segundo nivel, en cada sub grupo se utilizará la mediana de las Y's de dicho sub grupo. La construcción del árbol con n datos toma $O(nlogn)$.

En el principio se ha dicho que una ventaja de este tipo de árboles es que facilitan la búsqueda del vecindario dentro del radio de un punto, esto se consigue gracias al uso de una técnica de poda que permite descartar áreas completas (ramas) en la búsqueda, esta técnica aprovecha la forma en que se dividen los datos en cada nodo. En (b) de la imagen 3.5 se muestra un ejemplo de búsqueda dentro de un radio r, para saber cuales puntos están dentro se navega el árbol desde la raíz hasta el punto de consulta P calculando a su paso la distancia de cada punto visitado a P y reportándolo si la distancia es menor que r, como se puede observar, es posible el caso donde el radio de búsqueda pasa a otra región, sin embargo es fácil saber si se ha dado el caso y si no, el área se descarta por completo, el procedimiento para la técnica de poda es calcular la diferencia absoluta entre la componente que un nodo padre representa y esta misma componente en el punto P, si la diferencia es menor que r entonces el radio cruzó hacia esta área y será necesario realizar una búsqueda también dentro (utilizando también la técnica de poda en los subgrupos de esta región), en caso contrario toda la rama puede ser descartada. Por ejemplo, en el caso de la imagen el punto P es (7, 3) y visualmente podemos ver que la diferencia entre las componentes

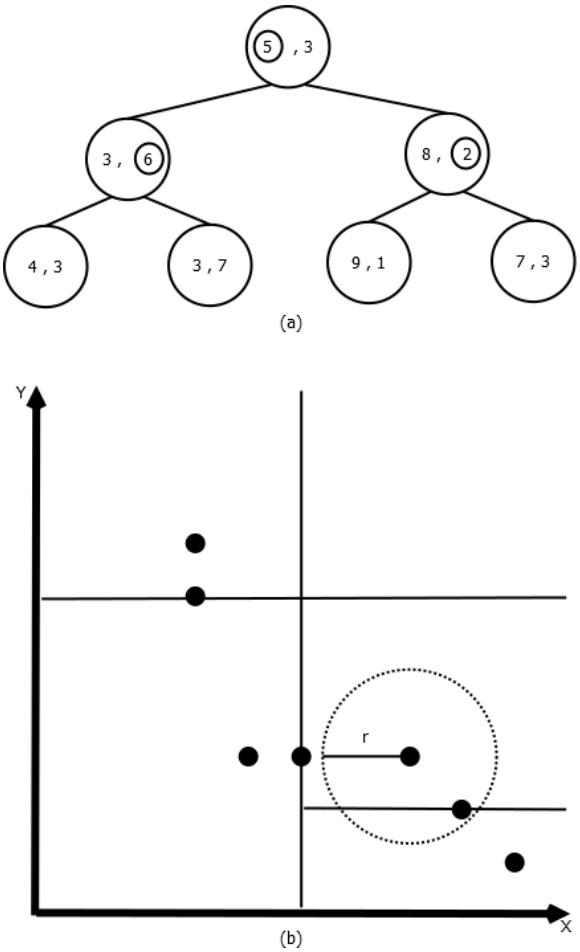


Figura 3.5: (a) Organización de los puntos {(5, 3), (3, 6), (8, 2), (4, 3), (3, 7), (9, 1), (7, 3)} en un K-d Tree y (b) divisiones correspondientes en el espacio 2 dimensional y un radio de búsqueda.

X que representa el primer nodo (5, 3) y la del punto de consulta es mucho mayor que r, es entonces imposible encontrar algún punto dentro del radio en la rama izquierda del nodo raíz y por lo tanto se descarta para búsqueda. Los algoritmos de construcción y búsqueda en k-d tree vienen implementados para 3 dimensiones en PCL (ver Apéndice A).

MLS Después de los filtros y la eliminación condicional de valores atípicos, se realizará una reconstrucción de la nube utilizando Moving Least Square, la razón de esta reconstrucción es obtener una nube de puntos ideal para calcular los descriptores sin que estos sean afectados por inconsistencias.

Como consecuencia de la reconstrucción, los datos quedarán libres del ruido que se mantuvo a pesar de los filtros y además rellenará aquellos huecos en donde el sensor no fue capaz de obtener información.

Este algoritmo se encuentra implementado en PCL (ver Apéndice A), a continuación se explicará de manera general para tener una idea más clara de su funcionamiento, si se desea información más detallada referirse a [28][29][30].

1. Sea P_C un punto cualquiera en la nube, se obtiene el vecindario V_C de este punto dentro de un radio de 10 mm.

2. Mover el centro de referencia de los puntos en V_C a P_C:

3. Estimar un plano tangente a P_C considerando V_C.

4. Asignar un valor escalar f_i a cada punto P_i en V_C, este valor corresponde con la distancia que cada punto tiene con el plano tangente.

5. Definir una función $\theta(d_{Ci})$ de ponderación que devuelve un valor más pequeño conforme la distancia es mayor entre el punto P_C y cada punto P_i en V_C. Esta función asigna más peso a los puntos más cercanos y menos a los que se encuentran más lejos del punto P_C, en la literatura existen diversas propuestas para esta función.

6. Ajustar una función polinomial que minimiza

$$\sum_i \theta(\| P_C - P_i \|) \| f(P_i) - f_i \|^2$$

donde $f(P_i)$ es la función polinomial. Esto se logra calculando:

$$f_{P_C}(P) = b(P)^T c(P_C) = b(P).c(P_C)$$

$$c(P_C) = \left[\sum_i \theta(d_{Ci}) b(P_i) b(P_i)^T \right]^{-1} \sum_i \theta(d_{Ci}) b(P_i) f_i$$

siendo b el vector de las bases del polinomio y c es el vector de coeficientes desconocidos que se busca minimizar. Por poner un ejemplo, los valores del vector b y c para un polinomio de grado 2 y dimensión espacial 2 son: $b(P) = \{1, x_p, y_p, x_p^2, x_p y_p, y_p^2\}$ y $c = \{c_{p1}, c_{p2}, c_{p3}, c_{p4}, c_{p5}, c_{p6}\}$.

7. Una vez calculada la función polinomial, se proyecta el punto P_C de la nube sobre la superficie definida por la función (Superficie MLS), se elimina el punto P_C y se genera un nuevo punto por la proyección. También fue posible proyectar algunos otros puntos del vecindario de P_C para conseguir una densidad mayor o rellenar huecos en la nube, en la implementación para este trabajo de tesis se estableció una densidad de al menos 25 puntos dentro del radio definido.

8. El procedimiento anterior completo se aplica sobre cada punto en la nube.

Con el algoritmo definido arriba se obtiene una nube libre de ruido sobre la cual se calcularán las normales y curvaturas tal como se cita a continuación.

3.6. Cálculo de normales

El Análisis de Componentes Principales ó PCA es un método estadístico que intenta explicar la estructura de la covarianza de los datos por medio de un número pequeño de componentes. En el enfoque clásico, la primer componente corresponde a la dirección en la cual los valores tienen la más grande varianza, la segunda componente es ortogonal a la primera y nuevamente representa la máxima varianza en esta dirección, igual para las siguientes componentes[32]. Cada componente en PCA está formada por un eigenvector (vector propio) que indica dirección y un eigenvalor (valor propio) que será la varianza en la dirección del vector propio. La idea desarrollada en esta sección consiste en encontrar con PCA en un vecindario local de puntos 3D la componente que representa la menor variabilidad (valor propio menor), el vector para esta componente aproximará la normal local, los dos componentes restantes definirán el plano tangente a la superficie [33]. En la literatura un valor propio a menudo se representan con λ_i y un vector propio con v_i, donde i es el número del componente.

Para el cálculo de las normales con la idea anterior se utilizó una versión de PCA robusta a ruido presentada en [31], la diferencia de esta propuesta con otras similares está en la función de ponderación, hace una repartición inversamente proporcional de la suma de las distancias a la media. En la práctica, esta versión de PCA demostró buenos resultados.

Los algoritmos 2 y 3 fueron implementados en c++, el primero recibe como entrada el vecindario de un punto dentro de un radio de 5 mm y devuelve 3 componentes PCA ordenadas con respecto a su factor de variabilidad (valor propio) de mayor a menor, el segundo busca un vecindario dentro de un radio de 5 mm para cada punto en la nube, utiliza el algoritmo 2 para obtener las componentes principales, extrae la normal y devuelve el conjunto de normales para la nube.

Algorithm 2 Análisis de Componentes Principales (PCA)

Require: Vecindario de un punto: BloquePuntos

Ensure: Componentes principales: ComponentesPrincipales

1: Media = CalculaMedia(BloquePuntos)
2: **for all** Punto en BloquePuntos **do**
3: Distancia = CalculaDistancia(Punto, Media)
4: SumaDistancias = SumaDistancias + Distancia
5: **end for**
6: **for** i = 1 hasta BloquePuntos.Tamaño() **do**
7: Distancia = CalculaDistancia(BloquePuntos(i), Media)
8: **if** Distancia = 0 **then**
9: Pesos(i) = 1
10: **else**
11: $Pesos(i) = \frac{1}{Distancia * SumaDistancias}$ {Función de pesos propuesta en [31]}
12: **end if**
13: SumaPesos = SumaPesos + Pesos(i)
14: SumaPuntosPesados = SumaPuntosPesados + (BloquePuntos(i) * Pesos(i))
15: **end for**
16: $MediaPesada = \frac{SumaPuntosPesados}{SumaPesos}$
17: **for** i = 1 hasta BloquePuntos.Tamaño() **do**
18: $MatrizCovarianza = MatrizCovarianza + ((BloquePuntos(i) - MediaPesada)(BloquePuntos(i) - MediaPesada)^T Pesos(i))$
19: **end for**
20: $MatrizCovarianza = \frac{MatrizCovarianza}{BloquePuntos.Tamano()}$
21: ValoresPropios = CalcularValoresPropios(MatrizCovarianza)
22: VectoresPropios = CalcularVectoresPropios(MatrizCovarianza)
23: ComponentesPrincipales = OrdenarPorValorDescentente(ValoresPropios, VectoresPropios) {Los 3 componentes se ordenan en una matriz de 3x4, cada columna contiene una componente, las tres primeras filas contienen los vectores propios y en la cuarta fila se encuentran los valores propios}
24: **return** ComponentesPrincipales

Algorithm 3 Cálculo de Normales con PCA

Require: Nube de puntos : NubePuntos

Ensure: Las normales de la nube de puntos : Normales

```
KdTreeNube = CreaKdTree(NubePuntos)
for i = 1 hasta NubePuntos.Tamaño() do
    Vecindario = EncuentraVecindario(KdTreeNube, NubePuntos(i), 5mm)
    if Vecindario.Tamaño() < 5 then
        Normales(i) = NULL {No hay suficientes puntos para calcular la normal.}
    else
        ComponentesPrincipales = PCA(Vecindario) {Algoritmo 2}
        Normal = ExtraerNormal(ComponentesPrincipales) {Ya que los componentes están ordenados por columna de mayor a menor en base a su valor propio, se extrae el vector propio de la última colummna, este aproxima la normal.}
        Normales(i) = Normal
        ... {Código para curvaturas, explicado en la sección 3.7}
    end if
end for
return Normales
```

3.7. Extracción de curvaturas

El cálculo de normales con PCA ha sido un paso importante para la extracción de curvaturas en esta sección. La elección de un descriptor se realizó tomando en cuenta su capacidad para representar correctamente una superficie local de puntos, su robustez en las variaciones de densidad, su facilidad para ser clasificadas como observaciones y la velocidad en el cálculo. Todas estas consideraciones son importantes debido a las condiciones de uso que tendrán, en la captura de datos no se puede garantizar una densidad de puntos adecuada, se tomará un vecindario local a cada punto dentro de un radio de 5 mm y la densidad de puntos dependerá de la distancia que el dispositivo de captura tenga del objeto, con esta información el descriptor deberá ser capaz de representar lo mejor posible el área local del vecindario, además de hacerlo simple para generar las observaciones de entrenamiento.

Las curvaturas son un tipo de descriptor que proporcionan suficiente información sobre la superficie, sin embargo, la mayoría de las técnicas de cálculo son lentas y requieren de una buena densidad de puntos cuando se trata de curvas muy pronunciadas. En [33] se propuso un método para representar eficientemente superficies basadas en puntos, utilizando para ello los valores propios de la matriz de covarianza de un vecindario con tamaño n para un punto p, el método es llamado "Surface Variation"(Variación de la superficie) y se define como una relación de los valores propios del siguiente modo:

$$\sigma_n(p) = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \quad \text{con } \lambda_0 \leq \lambda_1 \leq \lambda_2 \tag{3.11}$$

este descriptor, como se demostró en [33] está fuertemente relacionado con la curvatura de la superficie y por eso en esta sección se ha tratado como tal, pero es más adecuado que las técnicas clásicas cuando no existe buena densidad en la nube, incluso al aumentar el tamaño del vecindario los resultados se mantienen similares [34]. Ya se ha mencionado en la sección anterior, que los valores propios indican la varianza en las direcciones de mayor variación para un conjunto de datos, Surface Variation aprovecha esta información para describir la superficie con los puntos del vecindario local, ya que λ_0 describe la variación a lo largo de la normal, es decir, que tanto se desvían los puntos del plano tangente, un valor de 0 para $\sigma_n(p)$ indicaría que todos los puntos están sobre el plano tangente y el máximo valor de $\frac{1}{3}$ significa que los puntos están distribuidos de forma completamente isotrópica.

El cálculo del descriptor se puede hacer directamente a partir de los resultados obtenidos en el Análisis de Componentes Principales realizado para estimar la normal, solo será necesario realizar la operación definida en la ecuación 3.11. Para fines de una mejor comprensión, se muestra el algoritmo 4 con la modificación hecha al 3 para estimar y devolver las curvaturas.

Algorithm 4 Cálculo de Normales y Curvaturas con PCA

Require: Nube de puntos : NubePuntos

Ensure: Las normales y curvaturas de la nube de puntos : Normales, Curvaturas

```
KdTreeNube = CreaKdTree(NubePuntos)
for i = 1 hasta NubePuntos.Tamaño() do
    Vecindario = EncuentraVecindario(KdTreeNube, NubePuntos(i), 5mm)
    if Vecindario.Tamaño() < 5 then
        Normales(i) = NULL {No hay suficientes puntos para calcular la normal.}
    else
        ComponentesPrincipales = PCA(Vecindario) {Algoritmo 2}
        Normal = ExtraerNormal(ComponentesPrincipales) {Ya que los componentes están ordenados por columna de mayor a menor en base a su valor propio, se extrae el vector propio de la última columnna, este aproxima la normal.}
        Normales(i) = Normal
        Curvatura = CalcularCurvatura(ComponentesPrincipales) {Ecuación 3.11}
        Curvaturas(i) = Curvatura
    end if
end for
return Normales, Curvaturas
```

3.8. Generación de observaciones

Del cómputo de descriptores (Surface Variation o Curvaturas) conseguimos para cada punto en la nube un valor entre 0 y 0.3 que describe su superficie local. Para generar las observaciones que serán utilizadas en las siguientes etapas para el entrenamiento y reconocimiento, en esta sección se propone un método para clasificar las curvaturas basado en rangos de valores.

La idea consiste en dividir el rango de valores desde 0 a 0.3 en 12 secciones y clasificar cada curvatura en la sección que corresponda de acuerdo a su valor. Para la generación de observaciones se usará el alfabeto V= { C0, C1,

C2, C3, C4, C5, C6, C7, C8, C9, C10, C11 }.

Por comodidad se ha decidido multiplicar los valores de los descriptores por 1000 y eliminar los decimales, por lo tanto el rango a dividir será de 0 a 300 y la clasificación se ha realizado del siguiente modo:

- $curvatura = 0 \rightarrow C0$
- $0 < curvatura \leq 20 \rightarrow C1$
- $20 < curvatura \leq 40 \rightarrow C2$
- $40 < curvatura \leq 60 \rightarrow C3$
- $60 < curvatura \leq 80 \rightarrow C4$
- $80 < curvatura \leq 100 \rightarrow C5$
- $100 < curvatura \leq 120 \rightarrow C6$
- $120 < curvatura \leq 140 \rightarrow C7$
- $140 < curvatura \leq 160 \rightarrow C8$
- $160 < curvatura \leq 180 \rightarrow C9$
- $180 < curvatura \leq 200 \rightarrow C10$
- $curvatura > 200 \rightarrow C11$

Al finalizar la clasificación tendremos una secuencia de observaciones que han sido extraídas de la nube de puntos.

3.9. Entrenamiento

En la sección 2.3.6 se ha hablado de un algoritmo para el entrenamiento de los modelos ocultos de Markov, este algoritmo de nombre Baum-Welch basado en Expectation-Maximization (EM) fue descrito y utilizado tal como se detalla en el trabajo de Rabiner [6][22].

La implementación en C++ de Baum-Welch se realizó adaptando algunas estructuras de datos para representar las tablas de emisión, transición, valores α en forward y β en backward, estas estructuras de datos fueron

desarrolladas por Dekang Lin [URL9], además se ha seguido su esquema de archivos para el entrenamiento.

A continuación el algoritmo 5 se muestra como un resumen, éste procesa como entrada múltiples secuencias de observaciones generadas en 3.8, toma un modelo inicial propuesto y lo devuelve con las probabilidades de emisión y transición entrenadas de acuerdo a las secuencias de entrada. La imagen 3.6 representa el diagrama de bloques para la fase de entrenamiento.

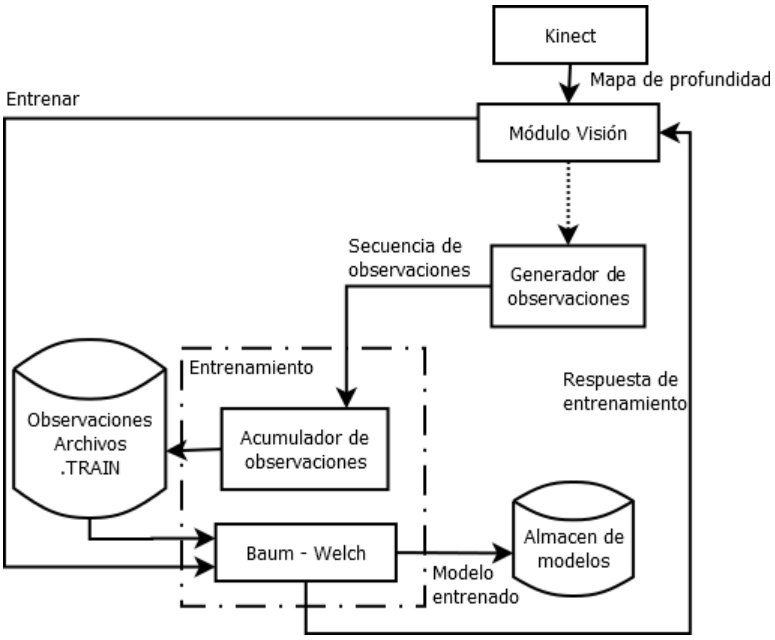


Figura 3.6: Diagrama de bloques del entrenamiento.

Una secuencia de observaciones completa se obtiene a partir de diversas tomas de un objeto segmentado sobre una superficie plana, en cada toma, dependiendo el tamaño del objeto, se consiguen un aproximado de 7500 observaciones, la cantidad de tomas deberá ser suficiente como para abarcar toda la superficie del objeto a entrenar y aportar información sobre las transiciones entre estados, experimentalmente se ha visto que un mínimo de 8 capturas es apto, sin embargo, más tomas lograrían un mejor entrenamiento, la imagen 3.8 muestra un ejemplo de 6 capturas para un objeto.

Para el modelo inicial se proponen dos diferentes arquitecturas con el objetivo de probar los resultados y la robustez con cada uno de ellas, la imagen 3.7 muestra los dos diseños. Como en la sección 2.3.3 se menciona,

estrictamente hablando los dos diseños son ergódicos pues desde cada nodo se puede llegar a cualquier otro en un número finito de pasos, la estructura lineal de los modelos se ha respetado en las dos propuestas debido a la forma en que se tiene pensado tomar las observaciones para realizar el entrenamiento, esto es, rotando el objeto sobre la mesa de acuerdo a las transiciones permitidas en el modelo inicial, en este sentido, la figura (a) representaría un giro de izquierda a derecha y la (b) permitiría el giro en cualquier sentido. En cada uno de los modelo propuestos, las probabilidades iniciales, de emisión y transición se distribuyeron equitativamente para la inicialización, es decir, la probabilidad inicial para cada estado es igual a $\left[\frac{1}{\text{Número estados}}\right]$, la probabilidad de emisión para un valor en un estado sería $\left[\frac{1}{\text{Número de valores a observar en el estado}}\right]$ y la probabilidad de transición de un estado i a uno j es $\left[\frac{1}{\text{Transiciones salientes del estado i}}\right]$, se ha considerado que todos los valores posibles de las observaciones se pueden observar desde cualquier estado. Las pruebas y el análisis de los resultados para cada tipo de modelo se presentan en el capítulo 5.

En este trabajo de tesis se han utilizado tres secuencias de observaciones para entrenar cada objeto, se sabe que entre más secuencias, mejor sería el entrenamiento, sin embargo cada secuencia contiene una cantidad de observaciones considerable lo cual se ha visto logra buenos resultados.

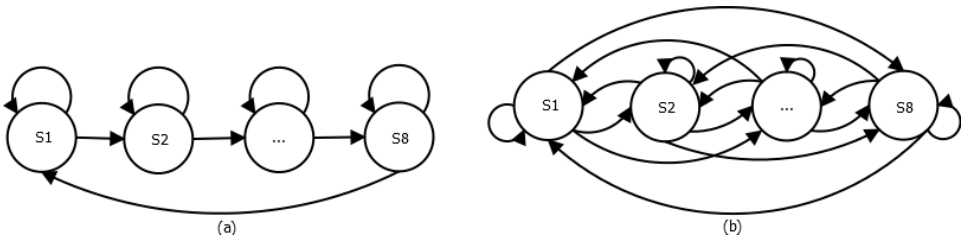


Figura 3.7: Modelos propuestos para representar los objetos.

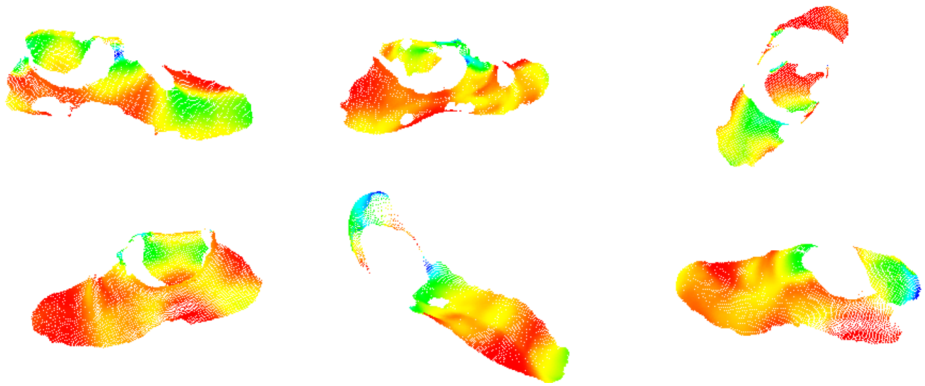


Figura 3.8: Diferentes capturas abarcando toda la superficie de un objeto.

Algorithm 5 Entrenamiento con Baum-Welch

Require: R secuencias de observaciones : O

Require: Modelo inicial : $\lambda = \{\Pi, A, B\}$

Ensure: Modelo entrenado : $\overline{\lambda} = \{\overline{\Pi}, \overline{A}, \overline{B}\}$

NUM_IT = 15
T = O.Tamaño()
N = Número de estados
$\alpha = Forward(O, \lambda)$ {Variable de Ec. 2.1 definida en sección 2.3.4}
$\beta = Backward(O, \lambda)$ {Variable de Ec. 2.5 definida en sección 2.3.4}
$ProbObs = P(O \mid \lambda) = \Pi_{r=1}^{R} P(O^r \mid \lambda) = \Pi_{r=1}^{R} \sum_{i=1}^{N} \alpha_{T_r}^r(i)$
for it = 1 hasta NUM_IT **do**
  for i = 1 hasta N **do**
    $\overline{\pi_i} = \sum_{r=1}^{R} \dfrac{\alpha_1^r(i)\beta_1^r(i)}{P(O^r \mid \lambda)}$
  end for
  for i = 1 hasta N **do**
    for j = 1 hasta N **do**
      $$\overline{a_{ij}} = \frac{\sum_{r=1}^{R} \sum_{t=1}^{T_r-1} \left[\dfrac{\alpha_t^r(i) a_{ij} b_j(O_{t+1}^r) \beta_{t+1}^r(j)}{P(O^r \mid \lambda)} \right]}{\sum_{r=1}^{R} \sum_{t=1}^{T_r-1} \left[\dfrac{\alpha_t^r(i)\beta_t^r(i)}{P(O^r \mid \lambda)} \right]}$$
    end for
  end for
  for j = 1 hasta N **do**
    $$\overline{b_j}(v_k) = \frac{\sum_{r=1}^{R} \sum_{t=1}^{T_r} \left[\dfrac{\left[\sum_{i=1}^{N} \alpha_{t-1}^r(i) a_{ij} b_j(v_k) \right] \beta_t^r(j)}{P(O^r \mid \lambda)} \right]}{\sum_{r=1}^{R} \sum_{t=1}^{T_r} \left[\dfrac{\alpha_t^r(j)\beta_t^r(j)}{P(O^r \mid \lambda)} \right]}$$
  end for
  $\alpha = Forward(O, \overline{\lambda})$
  $\beta = Backward(O, \overline{\lambda})$
  $NewProbObs = P(O \mid \lambda) = \Pi_{r=1}^{R} P(O^r \mid \lambda) = \Pi_{r=1}^{R} \sum_{i=1}^{N} \alpha_{T_r}^r(i)$
  if abs(ProbObs - NewProbObs) = 0 **then**
    return $\overline{\lambda} = \{\overline{\Pi}, \overline{A}, \overline{B}\}$
  else
    $\Pi = \overline{\Pi}$
    $A = \overline{A}$
    $B = \overline{B}$
    ProbObs = NewProbObs
  end if
end for
return $\overline{\lambda} = \{\overline{\Pi}, \overline{A}, \overline{B}\}$

3.10. Reconocimiento

Para la fase de reconocimiento se utilizó el algoritmo Forward, este se explica en la sección 2.3.4 y fue implementado de acuerdo a los trabajos de Rabiner [6][22].

Del mismo modo que en el entrenamiento, para la implementación de Forward en C++ se hizo uso de las estructuras desarrolladas por Dekang Lin [URL9].

Los algoritmos 6 y 7 se muestran a modo de resumen para comprender el proceso de evaluación de los diferentes modelos entrenados a partir de un conjunto de observaciones hechas, recibe como entrada una secuencia de observaciones, cada uno de los modelos entrenados y devuelve las probabilidades de que dichas observaciones pertenezcan a cada modelo, ordenadas de mayor a menor. La imagen 3.9 representa el diagrama de bloques para la fase de reconocimiento. La cantidad de tomas de observaciones necesarias que maximizan los mejores resultados será estimada experimentalmente en el capítulo 5.

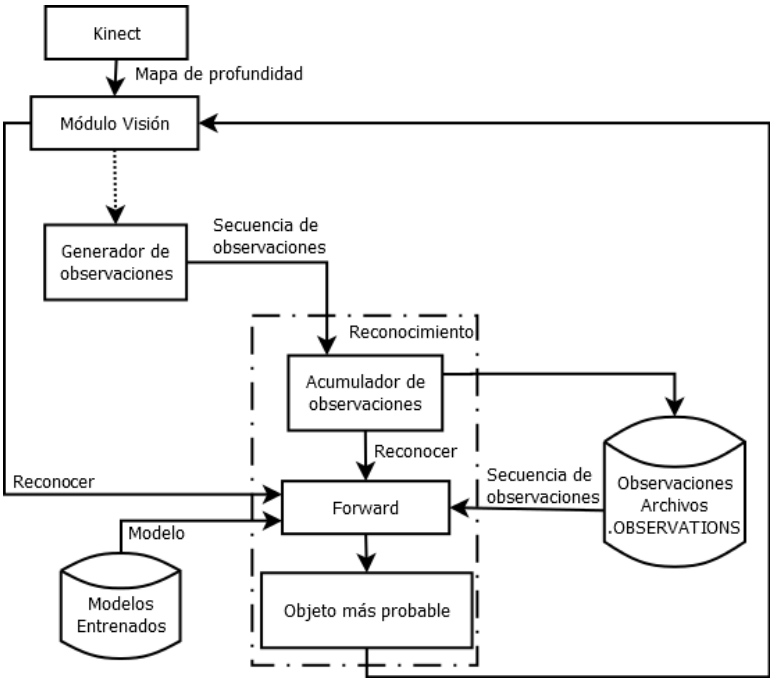


Figura 3.9: Diagrama de bloques del reconocimiento.

Algorithm 6 Evaluación con Forward

Require: Secuencia de observaciones : O

Require: Modelo a evaluar : $\lambda = \{\Pi, A, B\}$

Ensure: Probabilidad $P(O \mid \lambda)$: ProbObs

1: **for** i = 1 hasta N **do**
2: $\alpha_0(i) = 1$
3: $\alpha_1(i) = \pi_i b_i(o_1)$
4: **end for**
5: **for** t = 2 hasta T **do**
6: **for** j = 1 hasta N **do**
7: $\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij}\right] b_j(o_t)$
8: **end for**
9: **end for**
10: $ProbObs = P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i)$
11: **return** ProbObs

Algorithm 7 Reconocimiento usando Forward

Require: Secuencia de observaciones : O

Require: Modelos a evaluar : Mods

Ensure: Lista de objetos más probables ordenados descendente: ProbsObjsDesc

1: **for** i = 1 hasta Mods.Tamaño() **do**
2: ProbsObjs[i].Nombre = Mods[i].Nombre;
3: ProbsObjs[i].Prob = $Forward(O, Mods[i].\lambda)$
4: **end for**
5: ProbsObjsDesc = OrdernarDesc(ProbsObjs)
6: ProbsObjsDesc = SeleccionarMasProbables(ProbsObjsDesc)
7: **return** ProbsObjsDesc

3.11. Archivos de sistema y comandos de uso

En esta sección se explica el funcionamiento del sistema de reconocimiento a través de la descripción de los archivos necesarios y los comandos en el módulo de visión para la captura de observaciones, entrenamiento y reconocimiento.

3.11.1. Archivos de sistema

- Archivos .OBSERVATIONS: este tipo de archivos contienen una secuencia de observaciones separadas por espacios que serán utilizadas para la fase de reconocimiento.
- Archivos .TRAIN: contiene una secuencia de observaciones separadas por espacios que serán utilizadas para la fase de entrenamiento.
- Archivos .EMISSIONS: representan la tabla de probabilidades de emisión. Contienen la probabilidad de emisión para cada uno de los caracteres del alfabeto que es posible observar en cada estado.
- Archivos .TRANSACTIONS: son las probabilidades de transición entre estados.
- Archivo MODEL: este archivo contiene un listado con los nombres de aquellos modelos entrenados sobre los que se desea evaluar una secuencia de observaciones en la fase de reconocimiento.

3.11.2. Comandos del módulo de visión

- Comando CAPTURE [NOMBRE]: acumula observaciones en un archivo NOMBRE.TRAIN para entrenamiento, puede ser ejecutado tantas veces se desee, las observaciones nuevas se agregarán al final de las existentes si el archivo existe o creará uno nuevo en caso contrario.
- Comando TRAIN [NOMBRE]: realiza el entrenamiento de un modelo usando el archivo NOMBRE.TRAIN como secuencia de observaciones y los archivos GENERIC_MODEL.EMISSIONS y GENERIC_MODEL.TRANSACTIONS como modelo inicial. Genera los archivos NOMBRE.EMISSIONS y NOMBRE.TRANSACTIONS que serán los modelos entrenados con la secuencia de observaciones.
- Comando OBS [NOMBRE]: acumula observaciones en un archivo NOMBRE.OBSERVATIONS para reconocimiento, puede ser ejecutado tantas veces se desee, las observaciones nuevas se agregarán al final de las existentes si el archivo existe o creará uno nuevo en caso contrario. Cada vez que este comando se ejecuta guardará las nuevas observaciones adquiridas y las evaluará contra todos los modelos entrenados

que estén listados en el archivo MODELS. Devolverá la probabilidad de que las observaciones hayan sido generadas por cada uno de los modelos entrenados en orden descendente.

- Comando RECOG [NOMBRE]: realiza una evaluación de la secuencia de observaciones contenida en el archivo NOMBRE.OBSERVATIONS contra cada uno de los modelos entrenados listados en el archivo MODELS y devuelve las probabilidad de que las observaciones hayan sido generadas por cada uno de los modelos entrenados en orden descendente.

Capítulo 4

SISTEMA DE RECONOCIMIENTO HÍBRIDO PARA UN ROBOT DE SERVICIO

Uno de los objetivos del desarrollo de un sistema de reconocimiento utilizando modelos ocultos de Markov sobre una nube de puntos es mejorar el reconocimiento de objetos en los robots de servicio Judy y Justina desarrollados en el laboratorio de Biorobótica de la UNAM, la mejora en los resultados se piensa conseguir cuando los objetos a reconocer proporcionan más información por forma que por aspectos visuales en una imagen.

En el presente capítulo se describirá brevemente la arquitectura del módulo de visión, la interacción entre módulos del robot, la manera como este sistema propuesto se adapta al sistema de reconocimiento actual, se describirá también un poco de la técnica de reconocimiento vigente y el descriptor utilizado.

4.1. Arquitectura del módulo de visión

El módulo de visión es un sistema basado en comandos-respuestas que obtiene a través de un sensor kinect información del exterior y la utiliza para ejecutar tareas propias del área de visión computacional, fue desarrollado en el laboratorio de Biorobótica de la UNAM con el nombre de VisionStrikes-

Back para los robots de servicio que se construyen ahí mismo. La manera en que este módulo opera es recibiendo las peticiones de comandos que le llegan, ejecutando la tarea correspondiente y devolviendo una respuesta, estas peticiones se obtienen a través de un canal de comunicación desde un sistema de centralización tipo BlackBoard (sección 4.2) al cual es devuelta la respuesta. Dentro de las acciones que el módulo de visión puede ejecutar están:

- Reconocimiento de objetos por emparejamiento de descriptores locales.
- Detección de bordes en una mesa.
- Seguimiento de pinzas manipuladoras (gripper).
- Detección de personas.
- Reconocimiento de rostros.
- Segmentación de objetos sobre una mesa.
- Obtención de la posición de un objeto.

La imagen 4.1 es una representación gráfica de la arquitectura del módulo de visión. Ilustra la comunicación con el sistema Blackboard, con el sensor kinect y se representan con óvalos los diferentes comandos que puede ejecutar.

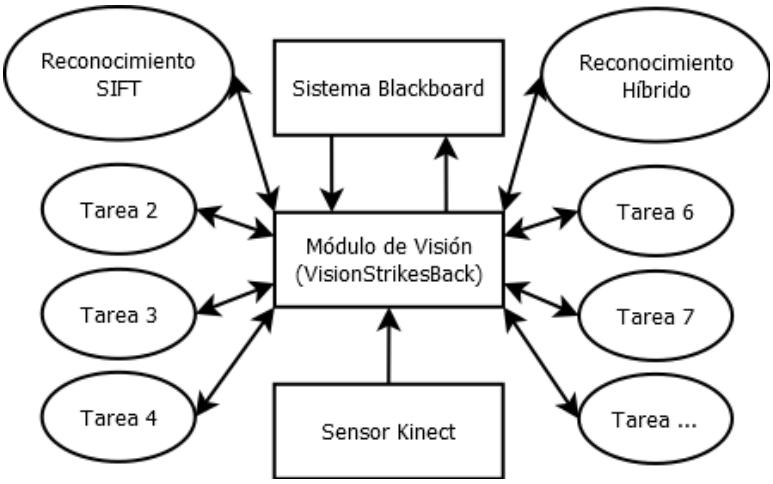


Figura 4.1: Arquitectura del módulo de visión.

4.2. Comunicación de los módulos del robot con sistema Blackboard

Una arquitectura Blackboard es una colección de programas independientes que trabajan cooperativamente sobre una estructura de datos en común. Cada programa está especializado en resolver una parte de la tarea completa y trabajan juntos para encontrar la solución. Estos programas especializados son independientes unos de otros. El componente Blackboard es el almacén de datos central donde los demás programas leen y escriben coordinados por un componente de control [36].

Los robots de servicio donde se implementará el sistema de reconocimiento usa una arquitectura tipo Blackboard cuyo componente central fue desarrollado por José Mauricio Matamoros de María y Campos para su tesis de maestría [35]. Como se ha definido antes, este tipo de arquitecturas facilitan la cooperación de diversos componentes o módulos para resolver tareas complejas, cada módulo se especializa en una tarea específica y se conecta al componente central Blackboard para conocer el estado general de la tarea, enterarse de las necesidades sobre las que puede cooperar y publicar sus resultados.

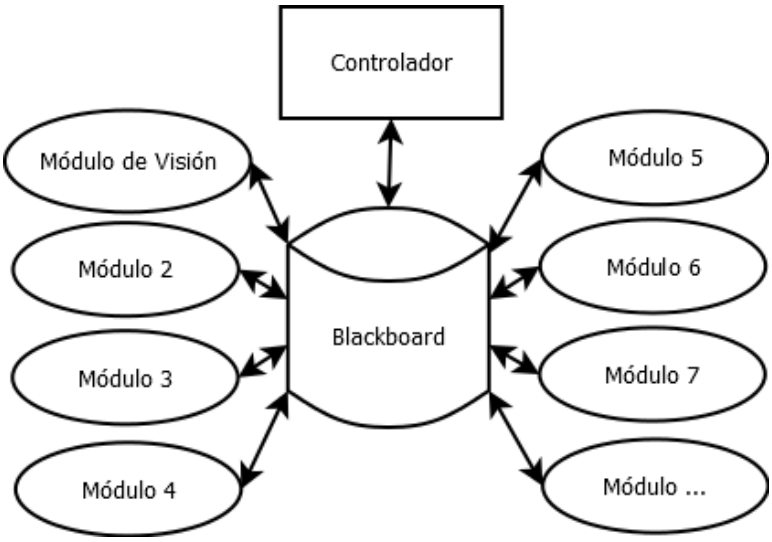


Figura 4.2: Arquitectura Blackboard.

La imagen 4.2 es diagrama representativo de una arquitectura Blackboard. El componente Blackboard es el centro de almacenamiento de datos,

este debe proveer una interfaz que permita a todos los demás programas leer y escribir en él; el controlador se encarga de orquestar el funcionamiento de los programas conectados (módulos) que trabajan cooperativamente en la solución y los módulos son aquellos programas especializados que resuelven tareas específicas con la información que consiguen del Blackboard y escriben de regreso una solución a estas tareas [36].

El módulo de visión es uno de estos programas especializados, su función es resolver las necesidades correspondientes a la comprensión del entorno a través de la visión, para lo cual posee implementados diversos algoritmos, algunos de ellos mencionados en la sección 4.1.

4.3. Propuesta de sistema de reconocimiento de objetos híbrido

La imagen 4.3 muestra un diagrama de bloques representando la manera en que se realiza la inclusión del sistema de reconocimiento por forma usando modelos ocultos de Markov al sistema de reconocimiento con SIFT que opera en Judy y Justina.

El sistema vigente usa solo la información de una imagen RGB para extraer descriptores locales SIFT y compararlos con aquellos entrenados anteriormente.

SIFT ó Scale-Invariant Feature Transform, es un algoritmo para describir una imagen con características invariantes a escala, fue propuesto por David G. Lowe en [37]. Básicamente este algoritmo busca puntos similares a esquinas en una imagen que sean lo suficientemente robustos como para ser localizados a diferentes niveles de escala, aplicando para ello diversas técnicas de evaluación que descarten a los más débiles, una vez localizados, se generará un descriptor por cada uno, considerando un vecindario de 16x16 para finalmente obtener un vector de 128 valores por descriptor, el total de descriptores en una imagen representan a la misma.

En un proceso de reconocimiento, los descriptores entrenados se hacen corresponder con aquellos obtenidos de una nueva captura, el emparejamiento descarta los falsos positivos usando una técnica de evaluación y devuelve las correspondencias fiables. El número de emparejamientos finales (al menos cuatro) dirá si las imágenes corresponden.

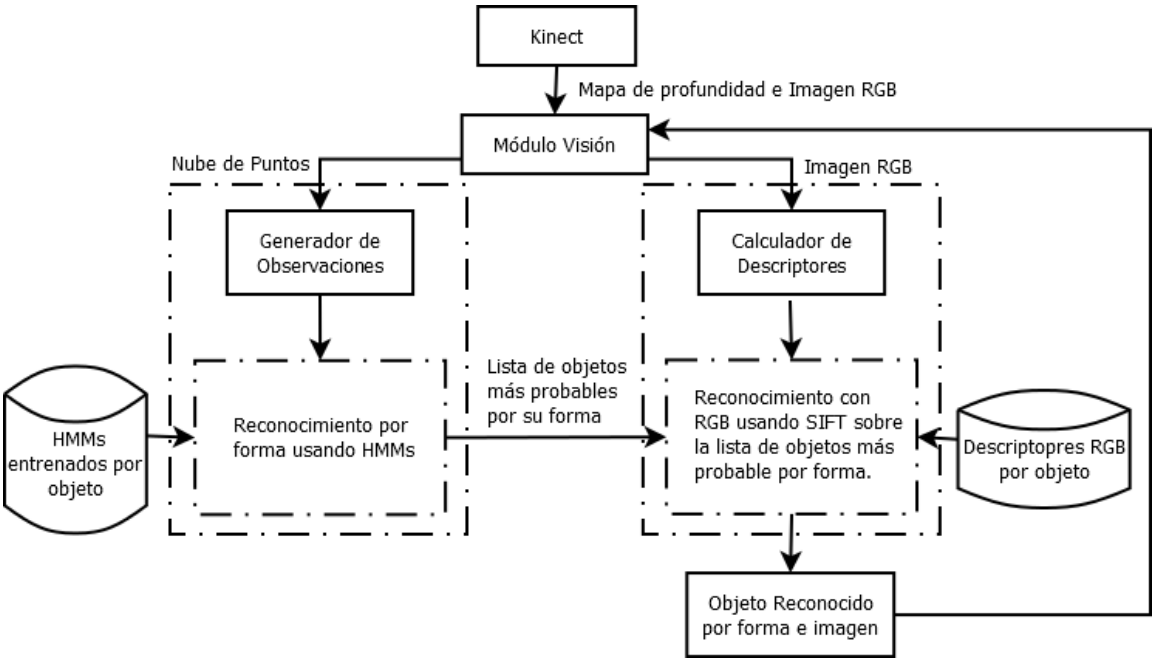


Figura 4.3: Diagrama de bloques de la adaptación del algoritmo de reconocimiento por forma usando HMMs con el sistema de reconocimiento actual en un robot de servicio.

A pesar de sus virtudes, reconocer con descriptores SIFT muestra debilidad cuando el objeto que se evalúa carece de los detalles que permiten generar los descriptores necesarios.

En esta propuesta, la idea consiste básicamente en aprovechar toda la información que el dispositivo Kinect nos puede proporcionar, tanto el mapa de profundidad como la imagen RGB. La adaptación del nuevo sistema de reconocimiento se propone como un filtro por forma como una fase previa al reconocimiento con SIFT, intentando de este modo fortalecerlo con información de la forma del objeto. El algoritmo completo que explica el funcionamiento de esta adaptación se describe:

1. Cada objeto entrenado consta de un nombre único, un modelo para reconocimiento con nube de puntos usando HMM y un conjunto de descriptores SIFT para el reconocimiento usando información RGB de la imagen.

2. Hecha una observación del objeto a reconocer, se obtiene la nube de puntos y la imagen correspondiente.

3. Con la nube de puntos se genera una secuencia de observaciones y se evalúa contra todos los modelos entrenados utilizando para ellos el algoritmo forward descrito en la sección 2.3.4.

4. Las probabilidades calculadas en el punto anterior se ordenan de mayor a menor.

5. Cada una de las probabilidades se normalizan en base a la probabilidad mayor usando la siguiente función:

$$ProbNorm = \frac{Prob}{ProbMayor} * 100$$

6. Se obtienen los nombres de aquellos objetos cuya probabilidad normalizada sea mayor a 90.

7. Utilizando el sistema de reconocimiento actual basado en descriptores SIFT sobre la imagen capturada, se realiza reconocimiento comparándola contra el conjunto de descriptores entrenados que correspondan a la lista de objetos devuelta en el punto anterior (con probabilidad normalizada mayor a 90).

8. Si el objeto ha sido encontrado con descriptores SIFT, se devuelve un valor true y el nombre del objeto reconocido, en caso de no ser encontrado, se devuelve un valor false y el nombre del objeto con mayor probabilidad según modelos ocultos de Markov.

Capítulo 5

PRUEBAS Y RESULTADOS

Durante este capítulo se realizarán las pruebas necesarias que revelen las ventajas y desventajas del reconocimiento de objetos utilizando información tridimensional y modelos ocultos de Markov. Dentro de las pruebas que se realizan está la prueba de reconocimiento con los dos diferentes tipos de HMM propuestos y la prueba comparativa entre el sistema de reconocimiento actual e híbrido implementados en el robot, finalmente se realizará una interpretación de los resultados obtenidos.

5.1. Pruebas

5.1.1. Objetos utilizados

Los objetos mostrados en el arreglo de imágenes 5.1 serán los utilizados para las pruebas presentadas en este capítulo, fueron seleccionados como un importante reto al algoritmo descrito en el actual trabajo de tesis. Muchos de los objetos seleccionados carecen de descriptores RGB pero poseen una forma bien definida, algunos otros comparten similitud en forma, lo cual representa una tarea desafiante para los algoritmos de reconocimiento basados en forma, como el evaluado aquí. La cantidad de objetos se eligió de acuerdo al máximo número de elementos utilizados en las pruebas de reconocimiento del torneo RoboCup donde se tiene pensado hacer uso del nuevo sistema.

Durante algunas pruebas, los objetos serán nombrados por el código que les ha sido asignado y en otras se utilizará el nombre corto.

Figura 5.1: Objetos para las pruebas.

Código	Nombre	Código	Nombre	Código	Nombre
SU	Suavitel	VC	VasoChico	TU	Tupper
CR	Crema	VG	VasoGrande	AG	Agarradera
LE	Leche	TZ	Taza	ZA	Zapato
CE	Cereal	PA	Papel	PL	Plancha
TA	Talco	TN	Tazón		
VM	VasoMediano	CA	Caja		

Tabla 5.1: Nombres de los objetos.

5.1.2. Pruebas de reconocimiento de objetos por tipo de HMM propuesto

Las pruebas planteadas en este apartado están enfocadas a comprobar la robustez, eficacia y eficiencia de las dos arquitecturas de la imagen 3.7. En cada caso se estima una tasa y tiempo promedio de reconocimiento correcto con diez muestras por objeto, además la matriz de confusión se presenta como herramienta para evaluar lo crítico de los errores en el proceso de reconocimiento.

El nuevo sistema de se ha evaluado realizando hasta 4 capturas por objeto desde diferente ángulo, esto con el objetivo de experimentar una de las ventajas que representa el uso de los modelos ocultos de Markov, al aumentar el número de observaciones hechas del fenómeno (objeto), se tiene más información para encontrar el modelo que las generó. Lo anterior se traduce en que entre más capturas se hagan de un objeto, más probable es reconocerlo correctamente. Una captura se puede definir como la toma de datos desde el dispositivo Kinect.

Para cada uno de los tipos de HMM se presentan dos tablas, las primeras (5.2 y 5.4) exponen las matrices de confusión y las siguientes (5.3 y 5.5) detallan la tasa y tiempo promedio de reconocimiento positivo por objeto así como la total, ya se ha dicho que para las pruebas se realizaron 10 eventos de reconocimiento por objeto, acumulando un máximo de hasta 4 capturas del objeto desde diferente ángulo, durante cada evento de reconocimiento o menos capturas en caso de lograr un resultado positivo antes, esto se explica mejor con un ejemplo extraído de la tabla 5.3, de los 10 eventos de reconocimiento hechos sobre el objeto "Suavitel", 20 % de las veces (2 eventos) bastó con una captura del objeto para que este fuera reconocido correctamente, 40 % de las veces (4 eventos) fueron necesarias 2 capturas, en 10 % de los eventos (1 evento) se necesitaron 3 capturas y para el 30 % (3 eventos) un reconocimiento adecuado se logró con 4 capturas, el "Suavitel", como se puede observar siempre fue reconocido correctamente, el "VasoGrande"por otro lado, el 10 % de las 10 veces no se logró reconocer de manera positiva ni con 4 capturas de este objeto.

Primer diseño propuesto de HMM

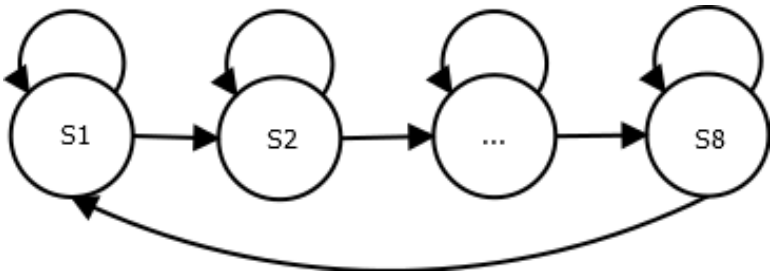


Figura 5.2: Primer diseño propuesto de HMM.

Matriz de confusión para primer tipo de HMM

Objetos reconocidos

Objetos a reconocer	SU	CR	LE	CE	TA	VM	VC	VG	TZ	PA	TN	CA	TU	AG	ZA	PL
SU	10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CR	-	10	-	-	-	-	-	-	-	-	-	-	-	-	-	-
LE	-	-	10	-	-	-	-	-	-	-	-	-	-	-	-	-
CE	-	-	-	10	-	-	-	-	-	-	-	-	-	-	-	-
TA	-	-	-	-	10	-	-	-	-	-	-	-	-	-	-	-
VM	-	-	-	-	-	10	-	-	-	-	-	-	-	-	-	-
VC	-	-	-	-	-	-	10	-	-	-	-	-	-	-	-	-
VG	-	-	-	-	-	-	-	9	-	-	1	-	-	-	-	-
TZ	-	-	-	-	-	-	-	-	10	-	-	-	-	-	-	-
PA	-	-	-	-	-	-	-	-	-	10	-	-	-	-	-	-
TN	-	-	-	-	-	-	-	-	-	-	10	-	-	-	-	-
CA	-	-	-	-	-	-	-	-	-	-	-	10	-	-	-	-
TU	-	-	-	-	-	-	-	-	-	-	-	-	10	-	-	-
AG	-	1	-	-	-	-	-	-	-	-	-	-	-	9	-	-
ZA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10	-
PL	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	9

Tabla 5.2: Matriz de confusión para primer tipo de HMM propuesto.

Resultados de reconocimiento para primer tipo de HMM

Objeto	Número de capturas necesarias para reconocer correctamente								
	1 Cap		2 Cap		3 Cap		4 Cap		% NR
	%	$\bar{t}(ms)$	%	$\bar{t}(ms)$	%	$\bar{t}(ms)$	%	$\bar{t}(ms)$	
Suavitel	20	862	40	2135	10	2970	30	3845	0
Crema	50	911	40	1676	0	0	10	4000	0
Leche	70	1035	20	2158	0	0	10	7581	0
Cereal	100	1357	0	0	0	0	0	0	0
Talco	90	802	10	1400	0	0	0	0	0
VasoMed	80	611	10	1895	10	1506	0	0	0
VasoChico	80	678	10	1601	10	2495	0	0	0
VasoGran	70	1151	10	1694	0	0	10	4911	10
Taza	100	579	0	0	0	0	0	0	0
Papel	100	844	0	0	0	0	0	0	0
Tazón	90	775	10	1512	0	0	0	0	0
Caja	100	1252	0	0	0	0	0	0	0
Tupper	80	1520	0	0	20	2071	0	0	0
Agarradera	50	1089	30	1893	10	3109	0	0	10
Zapato	40	2389	0	0	30	5757	30	8149	0
Plancha	80	2126	10	3276	0	0	0	0	10
PROM.	75	1088	11.875	1927	5.625	3499	5.625	5830	1.875

Tabla 5.3: Porcentaje y tiempo promedio de reconocimiento por capturas necesarias para 10 pruebas por objeto, utilizando el primer tipo de HMM propuesto. NR = No Reconocido

Segundo diseño propuesto de HMM

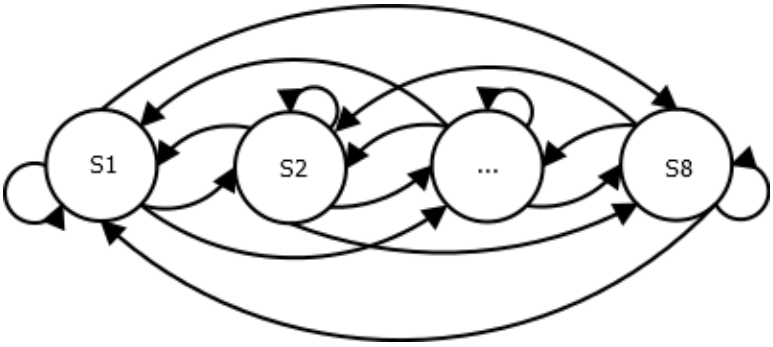


Figura 5.3: Segundo diseño propuesto de HMM.

Matriz de confusión para segundo tipo de HMM

Objetos reconocidos

Objetos a reconocer	SU	CR	LE	CE	TA	VM	VC	VG	TZ	PA	TN	CA	TU	AG	ZA	PL
SU	10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CR	-	10	-	-	-	-	-	-	-	-	-	-	-	-	-	-
LE	-	-	10	-	-	-	-	-	-	-	-	-	-	-	-	-
CE	-	-	-	10	-	-	-	-	-	-	-	-	-	-	-	-
TA	-	-	-	-	10	-	-	-	-	-	-	-	-	-	-	-
VM	-	-	-	-	-	10	-	-	-	-	-	-	-	-	-	-
VC	-	-	-	-	-	-	10	-	-	-	-	-	-	-	-	-
VG	-	-	-	-	-	-	-	9	-	-	1	-	-	-	-	-
TZ	-	-	-	-	-	-	-	-	10	-	-	-	-	-	-	-
PA	-	-	-	-	-	-	-	-	-	10	-	-	-	-	-	-
TN	-	-	-	-	-	-	-	-	-	-	10	-	-	-	-	-
CA	-	-	-	-	-	-	-	-	-	-	-	10	-	-	-	-
TU	-	-	-	-	-	-	-	-	-	-	-	-	10	-	-	-
AG	-	1	-	-	-	-	-	-	-	-	-	-	-	9	-	-
ZA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10	-
PL	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	9

Tabla 5.4: Matriz de confusión para segundo tipo de HMM propuesto.

Resultados de reconocimiento para segundo tipo de HMM

Objeto	Número de capturas necesarias para reconocer correctamente								
	1 Cap		2 Cap		3 Cap		4 Cap		% NR
	%	$\bar{t}(ms)$	%	$\bar{t}(ms)$	%	$\bar{t}(ms)$	%	$\bar{t}(ms)$	
Suavitel	20	2092	40	5070	10	7066	30	9045	0
Crema	50	2352	40	3969	0	0	10	10253	0
Leche	70	2517	20	5140	0	0	10	18989	0
Cereal	100	3261	0	0	0	0	0	0	0
Talco	90	2005	10	3446	0	0	0	0	0
VasoMed	80	1492	10	4597	10	3746	0	0	0
VasoChico	80	1761	10	3816	10	6043	0	0	0
VasoGran	70	2809	10	4113	0	0	10	12027	10
Taza	100	1385	0	0	0	0	0	0	0
Papel	100	2033	0	0	0	0	0	0	0
Tazón	90	1876	10	3825	0	0	0	0	0
Caja	100	3054	0	0	0	0	0	0	0
Tupper	80	3620	0	0	20	4954	0	0	0
Agarradera	50	2575	30	4492	10	6961	0	0	10
Zapato	40	5550	0	0	30	14010	30	19386	0
Plancha	80	5131	10	7692	0	0	0	0	10
PROM.	75	2638	11.875	4600	5.625	8417	5.625	14062	1.875

Tabla 5.5: Porcentaje y tiempo promedio de reconocimiento por capturas necesarias para 10 pruebas por objeto, utilizando el segundo tipo de HMM propuesto. NR = No Reconocido

Número de observaciones promedio por objeto

En la tabla 5.6 se muestra la cantidad de observaciones promedio obtenidas para realizar el correcto reconocimiento de cada objeto. Ya que las mismas observaciones se utilizaron para las pruebas en los dos tipos de HMM antes vistos, esta tabla aplica para ambos casos.

Objeto	Número de observaciones promedio
Suavitel	17234
Crema	10689
Leche	13470
Cereal	9557
Talco	6106
VasoMediano	5853
VasoChico	6727
VasoGrande	11555
Taza	4024
Papel	5913
Tazón	6034
Caja	8971
Tupper	11473
Agarradera	11133
Zapato	35691
Plancha	15848
PROMEDIO TOTAL	11237

Tabla 5.6: Número de observaciones promedio para lograr el reconocimiento.

5.1.3. Prueba comparativa entre sistema de reconocimiento actual e híbrido en el robot

El enfoque para esta prueba es comparar el cambio en la tasa de reconocimiento y en el tiempo promedio de respuesta para ambos sistemas, el actual usando SIFT y el nuevo utilizando HMMs. La adaptación a los robots del algoritmo presentado en esta tesis se realizó de acuerdo a lo descrito en el capítulo 4.

Recordando un poco sobre la implementación, existen dos etapas en la fase de reconocimiento, la primera basándose en información de profundidad para filtrar los objetos según su forma y la segunda haciendo uso de SIFT en la imagen para evaluar solamente contra aquellos elementos con más de 90 % de probabilidad en la primera etapa. La técnica seguida en la presente

prueba consiste en realizar hasta un máximo de cuatro capturas del objeto desde diferente ángulo y detenerse cuando SIFT ha logrado reconocer o quedarse con el de máxima probabilidad por forma al llegar a las cuatro capturas. La tabla 5.9 muestra los resultados en la comparación de los dos sitemas.

Matriz de confusión para sistema actual usando SIFT

Objetos a reconocer \ Objetos reconocidos	SU	CR	LE	CE	TA	VM	VC	VG	TZ	PA	TN	CA	TU	AG	ZA	PL	NR
SU	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3
CR	-	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1
LE	-	-	9	-	-	-	-	-	-	-	-	-	-	-	-	-	1
CE	-	-	-	9	-	-	-	-	-	-	-	-	-	-	-	-	1
TA	-	-	-	-	10	-	-	-	-	-	-	-	-	-	-	-	-
VM	-	-	-	-	-	10	-	-	-	-	-	-	-	-	-	-	-
VC	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10
VG	-	-	-	-	-	-	-	10	-	-	-	-	-	-	-	-	-
TZ	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10
PA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10
TN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10
CA	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	8
TU	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10
AG	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10
ZA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10
PL	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10

Tabla 5.7: Matriz de confusión para sistema actual usando SIFT. NR = No Reconocido

Matriz de confusión para sistema híbrido (SIFT y HMMs)

Objetos a reconocer \ Objetos reconocidos	SU	CR	LE	CE	TA	VM	VC	VG	TZ	PA	TN	CA	TU	AG	ZA	PL
SU	8	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CR	-	10	-	-	-	-	-	-	-	-	-	-	-	-	-	-
LE	-	-	10	-	-	-	-	-	-	-	-	-	-	-	-	-
CE	-	-	-	10	-	-	-	-	-	-	-	-	-	-	-	-
TA	-	-	-	-	10	-	-	-	-	-	-	-	-	-	-	-
VM	1	-	-	-	-	8	-	-	-	-	-	-	-	-	1	-
VC	-	-	-	-	-	-	10	-	-	-	-	-	-	-	-	-
VG	-	-	-	-	-	-	-	10	-	-	-	-	-	-	-	-
TZ	-	-	-	-	-	-	-	-	10	-	-	-	-	-	-	-
PA	-	-	-	-	-	-	-	-	-	10	-	-	-	-	-	-
TN	-	-	-	-	-	-	-	-	-	-	10	-	-	-	-	-
CA	-	-	-	-	-	-	-	-	-	-	-	10	-	-	-	-
TU	-	-	-	-	-	-	-	-	-	-	-	-	10	-	-	-
AG	2	-	-	-	-	-	-	-	-	-	-	-	-	8	-	-
ZA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10	-
PL	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10

Tabla 5.8: Matriz de confusión para sistema híbrido propuesto.

Resultados de comparación entre sistema actual con SIFT e híbrido (SIFT y HMMs)

Objeto	Porcentaje y tiempo de reconocimiento			
	Sistema Actual		Sistema Híbrido	
	%	$\bar{t}(ms)$	%	$\bar{t}(ms)$
Suavitel	70	940	80	1983
Crema	90	847	100	1176
Leche	90	920	100	1380
Cereal	90	966	100	1878
Talco	100	867	100	1069
VasoMediano	100	795	80	1195
VasoChico	0	0	100	949
VasoGrande	100	847	100	1241
Taza	0	0	100	964
Papel	0	0	100	1018
Tazon	0	0	100	960
Caja	20	690	100	1259
Tupper	0	0	100	1725
Agarradera	0	0	80	1198
Zapato	0	0	100	1911
Plancha	0	0	100	2581
PROM.	41.2	875.2	96.2	1403.7

Tabla 5.9: Tabla comparativa de porcentaje y tiempo promedio de reconocimiento entre sistema de reconocimiento actual e híbrido propuesto, para 10 pruebas por objeto.

5.2. Análisis e interpretación de resultados

Las pruebas anteriores fueron diseñadas para evaluar la eficacia y eficiencia de los modelos ocultos de Markov aplicados al reconocimiento de objetos usando su nube de puntos. Se buscó develar las fortalezas y debilidades en esta propuesta hasta ahora no estudiada a profundidad, para cada objeto fueron registrados los resultados con el objetivo de observar las dificulta-

des en el reconocimiento de acuerdo a las características de los mismos. A continuación se muestra a manera de puntos un análisis de los resultados:

- En los dos tipos de HMM propuestos se obtuvo la misma efectividad, la única diferencia se encontró en el tiempo necesitado para calcular los resultados, teniendo un incremento promedio del tiempo de 2.4 veces en el segundo tipo con respecto al primero.
- La tasa de reconocimiento correcto total utilizando únicamente el sistema con modelos ocultos de Markov sobre la nube de puntos fue 98.125 %.
- De 160 pruebas, el 75 % de las veces se reconoció el objeto correctamente utilizando solo una captura.
- De 160 pruebas, 11.875 % de las veces se necesitaron 2 capturas del objeto desde diferente ángulo para reconocer adecuadamente.
- De 160 pruebas, 5.625 % de las veces se necesitaron hasta 3 capturas del objeto desde diferente ángulo para reconocer adecuadamente.
- De 160 pruebas, 5.625 % de las veces se necesitaron hasta 4 capturas del objeto desde diferente ángulo para reconocer adecuadamente.
- Solo 1.875 % de las 160 pruebas de reconocimiento resultó en una confusión, tal como se detalla en la tablas 5.2 y 5.4 con los falsos positivos.
- Los resultados muestran que los objetos que fueron reconocidos con más facilidad (con solo una captura) son aquellos que desde cualquier ángulo tienen una vista similar y además no comparten demasiada apariencia con otros, para los artículos con diferente superficie en cada uno de sus lados se necesitaron hasta cuatro capturas de diferente ángulo para poder reconocer adecuadamente.
- En la tabla 5.6 se observa que la cantidad de observaciones tiene una relación directa con el tamaño de la superficie observada en una captura. En cada caso se asegura una densidad de 25 puntos dentro de un radio esférico de 1 cm en la superficie.
- El sistema de visión actual con SIFT tiene una efectividad de reconocimiento promedio de 91.42 % en objetos con imágenes en su superficie, mientras que tiene el 2.22 % sobre aquellos de color sólido.

- El sistema híbrido propuesto en el capítulo 4 aumentó la tasa de reconocimiento para los 16 objetos del apartado 5.1.1 en 133 % (de 41.2 % a 96.2 %), pero también aumentó 60 % el tiempo promedio de ejecución por objeto reconocido correctamente.

Capítulo 6

CONCLUSIÓN Y TRABAJO A FUTURO

A lo largo de este trabajo se descubrieron las ventajas y desventajas de los modelos ocultos de Markov en una aplicación poco explorada, el reconocimiento de objetos 3D usando nubes de puntos. En el presente capítulo se describirán las fortalezas y debilidades descubiertas para la propuesta hecha, también se habla de técnicas que reduzcan estas debilidades dentro del trabajo que se desea a futuro así como de otras mejoras e implementaciones. El objetivo en este capítulo es concluir si los objetivos se han cumplido satisfactoriamente y si la hipótesis resultó verdadera.

6.1. Conclusión

El uso de modelos ocultos de Markov en el reconocimiento 3D de objetos representó una propuesta eficaz cuando los objetos no poseen información visual suficiente, los resultados obtenidos demostraron que es posible mejorar la tasa de reconocimiento en elementos que presentan un reto para los algoritmos que hacen uso de descriptores locales en una imagen.

Lo crítico en el proceso completo del sistema recayó en la etapa de entrenamiento, es sabido que para obtener un modelo que mejor represente un fenómeno es necesario contar con suficientes muestras de éste, para entrenar adecuadamente los modelos ocultos de Markov con el algoritmo de Baum-Welch, como se mencionó en la sección 2.3.6, es necesario conseguir múltiples secuencias de observaciones, esto se logra fácilmente cuando se

trata de entrenar palabras para reconocimiento de voz, pero para objetos resulta en un trabajo tedioso, pues una buena secuencia se consigue realizando observaciones desde todos los ángulos visibles y se debería repetir este proceso suficientes veces para obtener tantas secuencias como sea posible y así entrenar el modelo.

Por otro lado, la nube de puntos capturada con el dispositivo Kinect fue de muy mala calidad, en un intento por mejorarla se realizó la reconstrucción de la misma, con esto se logró una nube libre de ruido y anomalías pero como consecuencia se perdieron detalles importantes de la superficie del objeto que pudieron diferenciarlos de otros fácilmente en la etapa de reconocimiento, tal es el caso del suavitel, la crema, la leche, la agarradera y el zapato, estos elementos en su superficie poseen pequeños detalles que al realizar la reconstrucción por interpolación polinomial desaparecieron, como segunda consecuencia, se hizo necesario hacer más capturas de estos objetos en la fase de reconocimiento para conseguir resultados adecuados. La reconstrucción fue de mucha ayuda con el ruido, pero las lecturas del kinect en ocasiones estuvieron lejos de representar adecuadamente la superficie de los elementos.

En lo que respecta a la fase de reconocimiento para el sistema que usa solo HMMs, es importante mencionar que una debilidad relevante es la necesidad de tener que hacer hasta cuatro capturas del artículo de diferente ángulo para maximizar la probabilidad de un resultado correcto, consiguiendo hasta 98.125 % de efectividad, pues con una sola captura solo se logra una tasa de 75 %. A pesar de parecer poco práctico, es esta la manera como los HMMs funcionan, entre más observaciones se tienen del fenómeno, mayor será la probabilidad de calcular el modelo correcto que las produce. Otra debilidad importante para la etapa de reconocimiento con este sistema, es que al intentar reconocer un objeto que no se encuentra en la base de conocimiento, el algoritmo forward no lo sabría, por la manera como funciona devolvería el objeto entrenado más parecido.

El sistema híbrido, además de buscar mejorar los resultados actuales cuando se usa solo SIFT, se propuso también como una opción para fortalecer las debilidades que se pensó podían resultar en el sistema de reconocimiento por forma usando HMMs, al tener dos objetos de aspecto similar, sin embargo, el sistema evidenció ser robusto a estas condiciones.

Finalmente, aún con los problemas de captura mencionados antes, los resultados dan validez a la hipótesis planteada al principio, el sistema híbrido

que se busca implementar en el robot demostró mejorar los resultados del reconocimiento en 133 %.

6.2. Trabajo a futuro

El siguiente listado detalla trabajo a futuro en forma de mejoras, oportunidades y aplicaciones de la propuesta original:

- Realizar la adquisición de la nube de puntos con un dispositivo Kinect v2, este hace un escaneo por tiempo de vuelo, logrando datos con mucho menos ruido que la versión anterior, además de proporcionar imágenes FULL HD. Al obtener menos ruido en la adquisición, se hace innecesaria la reconstrucción, de este modo se conservarán más los detalles originales de la superficie en los objetos.
- Mejorar la técnica de segmentación de los objetos, de tal modo que sea posible detectarlos en cualquier condición, no necesariamente sobre una superficie plana.
- Probar otras alternativas de descriptores para generar las observaciones en la nube de puntos.
- Obtener menos observaciones más representativas de la superficie.
- Realizar el entrenamiento con muchas más secuencias de observaciones.
- Hacer una investigación más profunda sobre el problema de los falsos positivos con objetos que no se encuentran entrenados e implementar una propuesta.
- Iniciar las probabilidades de emisión en cada estado del modelo inicial, utilizando una técnica de conteo estadístico con la primer secuencia de observaciones durante el entrenamiento.
- Implementar el sistema híbrido en un robot de servicio para ser utilizado en los torneos de RoboCup, Rockin y TMR.

Apéndice A

CÓDIGOS

A.1. Kd-Tree

El siguiente ejemplo fue extraído de [URL10]:

```
#include <pcl/point_cloud.h>
#include <pcl/kdtree/kdtree_flann.h>

#include <iostream>
#include <vector>
#include <ctime>

int
main (int argc, char** argv)
{
srand (time (NULL));

pcl::PointCloud<pcl::PointXYZ>::Ptr cloud (new pcl::PointCloud<
    pcl::PointXYZ>);

// Generate pointcloud data
cloud->width = 1000;
cloud->height = 1;
cloud->points.resize (cloud->width * cloud->height);

for (size_t i = 0; i < cloud->points.size (); ++i)
{
        cloud->points[i].x = 1024.0f*rand()/(RAND_MAX + 1.0f);
        cloud->points[i].y = 1024.0f*rand()/(RAND_MAX + 1.0f);
        cloud->points[i].z = 1024.0f*rand()/(RAND_MAX + 1.0f);
```

```
}

pcl::KdTreeFLANN<pcl::PointXYZ> kdtree;

kdtree.setInputCloud (cloud);

pcl::PointXYZ searchPoint;

searchPoint.x = 1024.0f * rand () / (RAND_MAX + 1.0f);
searchPoint.y = 1024.0f * rand () / (RAND_MAX + 1.0f);
searchPoint.z = 1024.0f * rand () / (RAND_MAX + 1.0f);

// Neighbors within radius search

std::vector<int> pointIdxRadiusSearch;
std::vector<float> pointRadiusSquaredDistance;

float radius = 256.0f * rand () / (RAND_MAX + 1.0f);

if ( kdtree.radiusSearch (searchPoint, radius,
    pointIdxRadiusSearch, pointRadiusSquaredDistance) > 0 )
{
        for (size_t i = 0; i < pointIdxRadiusSearch.size(); ++i)
        {
                std::cout << "    "
                << cloud->points[ pointIdxRadiusSearch[i] ].x
                << " "
                << cloud->points[ pointIdxRadiusSearch[i] ].y
                << " "
                << cloud->points[ pointIdxRadiusSearch[i] ].z
                << " (squared distance: "
                << pointRadiusSquaredDistance[i]
                << ")"
                << std::endl;
        }
}


return 0;
}
```

A.2. Moving Least Squares (MLS)

El siguiente ejemplo fue extraído de [URL11]:

```
#include <pcl/point_types.h>
#include <pcl/io/pcd_io.h>
#include <pcl/kdtree/kdtree_flann.h>
#include <pcl/surface/mls.h>

int main (int argc, char** argv)
{
        // Load input file into a PointCloud<T> with an
            appropriate type
        pcl::PointCloud<pcl::PointXYZ>::Ptr cloud (new pcl::
            PointCloud<pcl::PointXYZ> ());
        // Load bun0.pcd -- should be available with the PCL
            archive in test
        pcl::io::loadPCDFile ("bun0.pcd", *cloud);

        // Create a KD-Tree
        pcl::search::KdTree<pcl::PointXYZ>::Ptr tree (new pcl::
            search::KdTree<pcl::PointXYZ>);

        // Output has the PointNormal type in order to store the
             normals calculated by MLS
        pcl::PointCloud<pcl::PointNormal> mls_points;

        // Init object (second point type is for the normals,
            even if unused)
        pcl::MovingLeastSquares<pcl::PointXYZ, pcl::PointNormal>
             mls;

        // Set parameters
        mls.setInputCloud (cloud);
        mls.setPolynomialFit (true);
        mls.setSearchMethod (tree);
        mls.setSearchRadius (10);
        mls.setPointDensity(25);
        mls.setUpsamplingMethod(pcl::MovingLeastSquares<pcl::
            PointXYZ, pcl::PointNormal>::RANDOM_UNIFORM_DENSITY);

        // Reconstruct
        mls.process (mls_points);
```

```
        // Save output
        pcl::io::savePCDFile ("bun0-mls.pcd", mls_points);
}
```

Bibliografía

Libros y Artículos

[1] 50 Years of Object Recognition: Directions Forward, Alexander Andreopoulos, John K. Tsotsos, Computer Vision and Image Understanding, Volume 117, Issue 8, August 2013, Pages 827-891.

[2] Lawrence G. Roberts. Machine perception of three-dimensional solids. Thesis (Ph. D.) Massachusetts Institute of Technology, 1963.

[3] Visión por computador. Imágenes digitales y aplicaciones, Gonzalo Pajares Martinsanz, Jesús M. de la Cruz García, Alfaomega Ra-Ma, 2a Edición, Marzo 2008, ISBN: 978-970-15-1356-9, Capítulo 20: Descripción y Reconocimiento de Objetos 3D.

[4] Encyclopedia of Biometrics, Springer US, 2009, ISBN: 978-0-387-73002-8, Chapter: Deformable Models, Thomas Albrecht, Marcel Luthi and Thomas Vetter.

[5] Objet Recognition from Local Scale-Invariant Features, David G. Lowe, Computer Science Department, University of British Columbia, Vancouver, Canada. Published in: Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on.

[6] An Introduction to Hidden Markov Models, L.R. Rabiner, B.H. Juang, IEEE ASSP Magazine, January 1986.

[7] Object recognition using hidden Markov models, J. Hornegger, H. Niemann, D. Paulus and G. Schlottke, Pattern Recognition in Practice IV, p. 37 - 44, Vlieland, Netherlands, 1994.

[8] 3D Object Recognition Using Hidden Markov Models, Young Kug Ham, Kil Moo Lee, and Rae-Hong Park, Visual Conmmunications and Image Processing '95, 148 (April 21, 1995).

[9] 3D Object recognition in range images using hidden markov models and neural networks, Young Kug Ham, Rae-Hong Park, Pattern Recognition, Volume 32, Issue 5, May 1999, Pages 729-742.

[10] A Hidden Markov Model approach for appearance-based 3D object recognition, Manuele Bicego, Umberto Castellani, Vittorio Murino, Pattern Recognition Letters, Volume 26, Issue 16, December 2005, Pages 2588-2599.

[11] Hacking the Kinect, Jeff Kramer, Nicolas Burrus, Florian Echtler, Daniel Herrera C., Matt Parker. Editorial Apress, 2012. Distribuido por Springer Science+Business Media New York.

[12] Client Guide to 3D Scanning and Data Capture, Tristan Randall, David Philp. BIM Task Group, Julio 2013.

[13] 3D Laser Scanning for Heritage: Advice and guidance to users on laser scanning in archaeology and architecture (Second Edition), David Barber, Jon Mills. English Heritage, 2011.

[14] Evaluation of the spatial resolution accuracy of the face tracking system for Kinect for Windows V1 and V2, Clemens Amon, Ferdinand Fuhrmann. Alps-Adria Acoustics Assosiation, October 2014.

[15] S. Kashioka, M. Ejiri, Y. Sakamoto, A transistor wire-bonding system utilizing multiple local pattern matching techniques, IEEE Trans. Syst. Man Cybern. 6(8)(1976) 562-570.

[16] D. Marr, Vision: A Computational Investigation into the Human Representation and Processsing of Visual Information, W.H. Freeman and Company, 1982.

[17] J. Tsotsos, The Encyclopedia of Artificial Intelligence, chap. Image Understanding, John Wiley and Sons, 641 – 663, 1992.

[18] 3D is here: Point Cloud Library (PCL), Radu Bogdan Rusu and Steve Cousins. IEEE International Conference on Robotics and Automation (ICRA) 2011, Shanghai, China.

[19] Christian Wöhler. 3D Computer Vision: Efficient Methods and Applications. Second Edition, Springer, 2013.

[20] A. Ardeshir Goshtasby. 2-D and 3-D Image Registration for Medical, Remote Sensing, and Industrial Applications. Wiley-Interscience John Wiley & Sons, 2005.

[21] Richard Szeliski. Computer Vision: Algorithms and Applications. Springer, September 3, 2010.

[22] Lawrence R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. IEEE, Vol. 77, No. 2, February 1989.

[23] Horst Bunke, Terry Caelli. Hidden Markov Models Applications in Computer Vision. World Scientific, 2001.

[24] Francisco Javier Salcedo Campos. Modelos Ocultos de Markov: Del reconocimiento de voz a la música. Primera edición, Septiembre 2011.

[25] Martin A. Fischler, Robert C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Graphics and Image Processing, Volume 24, Number 6, June 1981.

[26] Ana Elizabeth García Hernández. Cálculo de varias variables. Grupo editorial Patria, S.A. de C.V., 2014.

[27] Jon Louis Bentley. Multidimensional Binary Search Trees Used for Associative Searching. Communications of ACM, Volume 18, Number 9, September 1975.

[28] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, Claudio T. Silva. Point Set Surfaces. VIS '01 Proceedings of the conference on Visualization '01, IEEE Computer Society Washington, 2001-10-21, pag. 21-28.

[29] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, Claudio T. Silva. Computing and Rendering Point Set Surfaces. Visualization and Computer Graphics, IEEE Transactions on (Volume:9 , Issue: 1), Jan-March 2003, pag. 3-15.

[30] Andrew Nealen. An As-Short-As-Possible Introduction to the Least Squares, Weighted Least Squares and Moving Least Squares Methods for Scattered Data Approximation and Interpolation. Discrete Geometric Modeling Group, TU Darmstadt

[31] Esmeide A. Leal Narváez, Nallig Eduardo Leal Narváez. Point Cloud Denoising Using Robust Principal Component Analysis. Proceedings of the First International Conference on Computer Graphics Theory and Applications, 2006, pages 51-58.

[32] Mia Hubert, Peter J. Rousseeuw, Karlien Vanden Branden. A New Approach to Robust Principal Component Analysis. American Statistical Association and the American Society for Quality TECHNOMETRICS, February 2005, VOL. 47, NO. 1.

[33] Mark Pauly, Markus Gross, Leif P. Kobbelt. Efficient Simplification of Point-Sampled Surfaces. Visualization, 2002. VIS 2002. IEEE, November 2002, Pag. 163 - 170.

[34] Mark Pauly, Richard Keiser, Markus Gross. Multi-scale Feature Extraction on Point-Sampled Surfaces. EUROGRAPHICS Volume 22 (2003), Number 3.

[35] José Mauricio Matamoros de María y Campos. Análisis de extensibilidad, reestructuración y desempeño de software para robots móviles. Tesis de maestría, IIMAS-UNAM, enero 2013.

[36] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerland and Michael Stal. Pattern-Oriented Software Architecture. A system of Patterns. Chichester, United Kingdom, John Wiley & Sons, 1996.

[37] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision, 2004.

Páginas Web

[URL1] https://msdn.microsoft.com/en-us/library/jj131033.aspx

[URL2] https://github.com/OpenNI/OpenNI/blob/master/ Documentation/OpenNI_UserGuide.pdf

[URL3] http://structure.io/openni

[URL4] http://docs.opencv.org/

[URL5] https://github.com/Itseez/opencv/wiki

[URL6] http://docs.pointclouds.org/trunk/

[URL7] http://docs.pointclouds.org/trunk/group__keypoints.html

[URL8] http://docs.pointclouds.org/trunk/group__features.html

[URL9] https://webdocs.cs.ualberta.ca/ lindek/hmm.htm

[URL10] http://pointclouds.org/documentation/tutorials/kdtree_search.php

[URL11] http://pointclouds.org/documentation/tutorials/resampling.php