UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

DETECCIÓN Y RECONOCIMIENTO DE OBJETOS USANDO IMÁGENES RGB Y NUBES DE PUNTOS ORGANIZADAS

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN CIENCIAS (COMPUTACIÓN)

PRESENTA:
JESUS CRUZ NAVARRO

TUTOR:
DR. JESÚS SAVAGE CARMONA
FACULTAD DE INGENIERÍA, UNAM

MÉXICO, D. F., FEBRERO 2016

Agradecimientos

... a Ari, mi hermanita, por todos estos años de amistad y cariño desmedido.

... a Celia, por su amor maternal.

... A mi padre, a Dafne y a Emily por su apoyo incondicional.

... al Dr. Jesus Savage, por todos sus consejos y paciencia.

... a los compañaeros del laboratorio, por sus opiniones e ideas.

... a los sinodales, por el tiempo dedicado a mejorar este trabajo.

... a CONACYT y a la DGAPA-UNAM a través del proyecto PAPIIT IG100915, por el apoyo brindado durante la realización de este trabajo.

Resumen

Existen numerosas técnicas para detectar y reconocer objetos, sin embargo, la mayoría se basan en la extracción y descripción de puntos de interés, los cuales, son difíciles de obtener si los objetos a reconocer tienen poca o nula textura.

En el presente trabajo se propone un sistema para la detección, segmentación y reconocimiento de objetos que se encuentran sobre planos horizontales, utilizando la información que proporciona una cámara RGB-D, es decir, la imagen de color y la nube de puntos de organizada de una escena. La detección y segmentación se realizan utilizando únicamente la información de profundidad, mientras que, para el reconocimiento, se procesa la imagen de color y la nube de puntos organizada de cada objeto detectado para caracterizarlo con base en su tamaño, forma y color. El sistema se probó experimentalmente obteniendo buenos resultados en cuanto a tiempos de ejecución y tazas de reconocimiento.

Los algoritmos desarrollados serán integrados al módulo de visión de Justina, un robot móvil de servicio desarrollado en la Facultad de Ingeniería de la UNAM, con el objetivo de ser utilizados en competencias internacionales de Robótica.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La idea de un robot que ayude a realizar tareas comunes en ambientes cotidianos, como puden ser una casa o una oficina, ha existido en la mente colectiva de las personas desde hace mucho tiempo. Basta con leer clásicos de la literatura de ciencia ficción o ver alguna película con tintes futuristas para darse cuenta de esto.

Una de las principales capacidades que debe tener un robot de servicio es la de detectar y reconocer diferentes objetos que se encuentren en su entorno. Ya sea para manipularlos, es decir, tomarlos y realizar alguna acción con ellos, o simplemente para obtener información y generar una base de conocimientos de lo que se encuentra a su alrededor.

La detección se refiere al proceso de decidir si existe o no un objeto en una escena, mientras que en el reconocimiento[1] se verifican las características de un objeto para decidir si corresponden a una instancia en particular. Estas características pueden venir de un modelo matemático o de un entrenamiento previo en el que se extraen y almacenan dichas características para compararlas posteriormente.

La detección y el reconocimiento de objetos no son temas nuevos. Actualmente, muchos procesos industriales utilizan estas técnicas para automatizar sus

[1]En algunos textos, al reconocimiento también se le denomina como identificación.

sistemas de producción. Un ejemplo de esto son las bandas transportadoras en las que, utilizando una serie de sensores de presencia y cámaras RGB, clasifican y separan diversos tipos de productos.

Sin embargo, para un robot de servicio, existen otro tipo de retos en cuanto a la detección y reconocimiento de objetos que no se encuentran comúnmente en los sistemas industriales, siendo el principal un ambiente no controlado.

En un ambiente no controlado, las condiciones de luz, la posición de los objetos, el ángulo de visión de la cámara, el contexto de la escena detectada y otros parámetros pueden variar para diferentes capturas del sensor, por lo que los algoritmos de detección y reconocimineto tienen que ser roboustos a iluminación, escala, rotación y perspectiva de los objetos, cambios en el fondo de la escena, etc.

Actualmente, existen una gran variedad de técnicas con buenos resultados para reconocer objetos en una imagen RGB, obtenida bajo condiciones no controladas. Estas se basan en la detección y descripción de puntos característicos en regiones de los objetos con cambios distintivos en la intensidad de los pixeles, como son manchas, bordes, textura, etc. Sin embargo, cuando se aplican estas técnicas en objetos con poca o nula textura, estos son poco confiables.

En los ultimos años, gracias en gran parte al desarrollo de sensores de bajo costo, como el Kinect, nace la tendencia a utilizar cámaras RGB-D en la robótica, las cuales, además de información de color, proporcionan información de profundidad en una escena. La ventaja de utilizar este tipo de sensores es evidente: se puede extraer mucha más información que con un sensor dedicado a extraer sólo información de color. Además, en un robot de servicio, este tipo de cámatas, pueden serivir no sólo para las tareas de detección y reconocimiento de objetos, sino como datos de entrada para desemepeñar otras funciones importantes como pueden ser la localización y mapeo simultáneos o el reconocimiento de personas y acciones, etc.

En el presente trabajo, se pretende abordar la detección y el reconocimiento de objetos utilzando técnicas que no estén basadas en la detección de puntos característicos, sino en tres características que poseen todos los objetos que se

encuentran a nuestro alrededor: el tamaño, la forma y el color. Para esto, se utilizó la información que proporciona una cámara RGB-D, la cual forma parte del sistema de sensores del robot Justina, un robot móvil de servicio desarrollado en el Facultad de Ingeniería de la UNAM.

1.1. Objetivo

El objetivo general del presente trabajo es:

> Desarrollar un sistema que permita, de manera eficiente, detectar y reconocer objetos con poca o nula textura[2], que se encuentren sobre planos horizontales en una escena, utilizando la imagen RGB y la nube de puntos que proporciona el sensor Kinect del robot Justina. Los objetos a ser reconocidos deben de haber sido entrenados previamente.

Para cumplir con este objetivo, se plantearon las siguientes actividades:

- Diseñar un algoritmo que permita obtener los planos horizontales de una escena, como pueden ser mesas, libreros, estantes, etc.
- Diseñar e implementar un algoritmo que permita obtener la nube de puntos y la imagen RGB de los objetos que se encuentren sobre los planos horizontales, es decir, detectar y segmentar cada objeto.
- Diseñar un algoritmo que permita extraer información sobre la forma, tamaño y color de los objetos detectados e identifique de qué objeto se trata.
- Desarrollar un sistema que utilice los algoritmos diseñados y que se integre a la arquitectura actual del robot Justina para su utilización en compentencias posteriores. Además, este sistema debe contar con un entrenador de objetos que permita generar la base de objetos a reconocer.

[2]Si un objeto tiene poca o nula textura, será dificil obtener puntos de interes, en los que se basan la mayoría de las técnicas de reconocimiento utilizadas actualmente.

- Generar una metodología para el uso del sistema desarrollado y probarlo experimentalmente.

1.2. Hipótesis

La hipótesis que se generó al iniciar el desarrollo de este trabajo y en la cual se basa el objetivo del mismo es:

> Se consiguen mejores resultados al identificar un objeto con poca o nula textura analizando sus dimensiones, la forma que proyecta este en el plano en el que se encuentra suspendido y su patrón de color, que utilizando técnicas basadas en la descripción de puntos característicos.

Para concluir este trabajo y validar la hipotesis mencionada anteriormente, se desarrollaron una serie de pruebas experimentales.

1.3. Motivación

Desde el 2011, en el Laboratorio de Biorobótica, ubicado en la Facultad de Ingeniería de la UNAM, se viene desarrollando el proyecto Justina, a cargo del Dr. Jesus Savage Carmona, el cual, tiene como objetivo desarrollar un robot de servicio que ayude a las tareas del hogar.

En el 2014, para probar el desempeño del robot, el equipo asistió a la competencia de robótica RoCKIn@Home (RoCKIn, 2015) para robots de servicio. En esta competencia, existen una seria de pruebas, denominadas *Functionality Benchmarks*, diseñadas para probar capacidades individuales del robot.

Una de estas pruebas, denominada *Object Perception*, trataba de evaluar la capacidad del robot de procesar y analizar información sobre una lista de objetos observados. Esta prueba requería que el robot detectara la presencia de los objetos, los categorizara, identificara su instancia y estimara su posición. Asimismo,

Figura 1.1: Lista de objetos utilizados durante la competencia RoCKIn 2014.

se consideraba el tiempo de reconocimiento de cada objeto para determinar la puntuación de cada equipo.

El sistema de visión utilizado en ese momento basaba la detección y reconocimiento de objetos en la busqueda y descripción de puntos de interes mediante el método SIFT, propuesto por Lowe (1999). Este sistema había sido utilizado en compentencias anteriores, exhibiendo un desempeño aceptable, aunque lento, al reconocer objetos con muchísima textura, por ejemplo, cajas de galletas o cereal, latas de refrescos, envases de bebidas energizantes, por mencionar algunos, en los que sus diseños son llamativos y contienen muchas imágenes y textos de diferentes colores y contrastes.

Sin embargo, los objetos que se necesitaban reconocer durante la prueba mencionada durante la competencia de RoCKIn del 2014 eran muy diferentes a los que se utilizaban comúnmente para probar el sistema. Se trataban de 10 objetos con colores sólidos y muy poca textura, los cuales se pueden ver en la figura 1.1. Debido a esto, lo resultados no fueron los esperados y apresuradamente se cambió el método de reconocimiento a uno basado en el color de los objetos. Esto mejoró los resultados de la prueba, sin embargo, no se obtuvo un lugar remarcable.

Partiendo de esta experiencia y para continuar con el trabajo iniciado durante la compentencia, nace la necesidad de desarrollar un sistema que permita al robot Justina detectar y reconocer objetos con poca o nula textura, pero con una forma, dimensiones y color distintivos.

1.4. Descripción del documento

El presente trabajo es resultado de la investigación realizada en el Laboratorio de Biorobótica en la Facultad de Ingeniería de la UNAM. Se encuentrá estructurado de forma secuencial, siguiendo los pasos que se fueron desarrollando durante la elaboración del mismo. Al final, se encuentra un apartado de apéndices, donde se presentan partes del código que integran el sistema desarrollado.

En el Capítulo 2 se presentan los conceptos téoricos y definiciones que dan sustento al desarrollo del trabajo. Asimismo, se expone un panorama general de las propuestas más relevantes en el área del detección y reconocimiento de objetos, tanto en 2D como 3D.

El Capítulo 3 trata sobre los algoritmos utilizados para la detección de objetos utilizando la nube de puntos organizada proporcionada por el Kinect. Iniciando con el preprocesamiento de estos datos, se sigue con la obtención de planos horizontales usando una adecuación del algoritmo RANSAC. Una vez obtenidos los planos horizontales, se obtienen los puntos corresponientes a los objetos y se segmentan para obtener una nube de puntos para cada objeto en los planos.

En el Capítulo 4 se habla de las técnicas utilizadas para la identificación de objetos. En la primera parte se explica la obtención y comparación del patrón de forma y tamaño usando información 3D de los objetos, mientras que en la segunda parte se utiliza la información de color de cada objeto en forma de histogramas. Para finalizar, se habla sobre la integración de estos dos características para clasificar correctamente un objeto.

En el Capítulo 5 se muestran los resultados obtenidos de manera experimental utilizando el sistema desarrollado y se analizan los mismos. Además, se realiza

una comparación entre el sistema propuesto y el sistema actual del robot Justina basado en SIFT.

Finalmente, en el Capítulo 6 se exponen las conclusiones obtenidas a lo largo del proceso de investigación surgido a partir de este trabajo. Además, se plantean ideas sobre el trabajo a seguir para continuar mejorando el sistema desarrollado.

Capítulo 2

Marco teórico

2.1. El sensor Kinect

El sensor Kinect fue desarrollado por la empresa Microsoft. Fué puesto a disposición del público en noviembre del 2010, como un dispositivo enfocado principalmente a la interacción natural en videojuegos, mediante poses y movimientos del cuerpo de los usuario.

Sin embargo, al ser este un sensor de muy bajo costo (comparándolo con otros sensores que proveen información similar, como pueden ser los escáneres láser), rápidamente atrajo la atención de muchos investigadores en el área de robótica. En la actualidad, es común ver robots que utilizan este tipo de sensor para desempeñar diversas tareas, como puede ser la de localización y mapeo simultáneo (SLAM) (Endres et al., 2012).

Muchos de los detalles del funcionamiento de este sensor no se conocen con exactitud, dado que se consideran un secreto comercial. Sin embargo, se sabe que para obtener el mapa de profundiad utiliza una técnica similar a la de proyección de luz estructurada. Esta técnica consiste básicamente en proyectar un patrón conocido de puntos de luz infrarroja para posteriomente capturar el cambio que se genera en dicho patrón debido a las superficies de los objetos en los que incide.

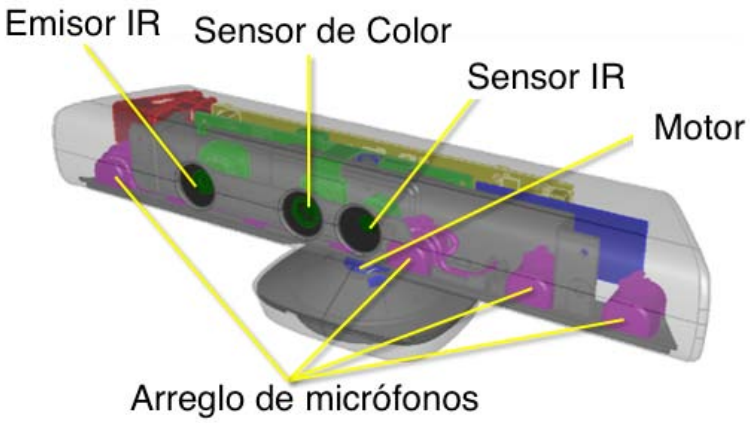


Figura 2.1: Componentes internos del sensor Kinect.

Analizando este cambio, se puede obtener una imagen de la profundidad de los objetos al plano de la cámara.

Una vez obtenida la información de profundidad, esta se relaciona con la información obtenida por lo cámara RGB para obtener imágenes de 4 canales. Estos son, tres con la información de color (RGB) y uno con información de profundidad (D). Con la información de profundiad y las parámetros de la cámara, se puede calcular la posición tridimensional de cada pixel de la imagen con el origen referenciado al centro de la cámara del sensor.

Además de los componentes que permiten construir las imagenes RGB-D, los cuales son una cámara de color, un emisor infrarrojo y un sensor infrarrojo, el sensor kinect cuenta con un arreglo de 4 microfonos y un motor que le permite girar con un ángulo de cabeceo en un rango de 54 grados. En la figura 2.1 se puede ver la dispocisión de dichos componetes en el sensor.

En la tabla 2.1 se enumeran las características de las imágenes de color y profundidad (Microsoft, 2010). Además, se especifíca la distancia máxima y mínima para poder obtener información de profundiad de un objeto.

La imagen de profundidad que genera el sensor Kinect comúnmente presenta datos sin información, es decir, pixeles en donde el valor de profunidad es 0, los cuales representan error en la lectura. Existen diversas fuentes para este tipo de

Característica	Valor
Ángulos de visión	43º en vertical y 57º en horizontal
Resolución máx de color	1280 x 960
Resolución máx. de profundidad	640 x 480
Rango de profundidad	0.8 - 4.0 metros
Frecuencia máx.	30 cuadros por segundo

Tabla 2.1: Características del sensor Kinect.

errores. A continuación se describen algunas de las más importantes (Khoshelham, 2011):

Condiciones de luz En escenas con luz muy intensa, se presentan errores en la deteccion del patrón de puntos proyectado, lo que da origen a regiones sin información o errores en la medición.

Distancia a los objetos Si los objetos se encuentran fuera del rango de medición del sensor, no se obtendrá el valor de profundiad de estos. Igualmente si la distancia esta fuera del rango, pero se tiene lectura, esta será muy ruidosa.

Distancia entre el sensor IR y el emisor IR Dado que el sensor IR y el emisor IR se encuentran físicamente separados por una distancia considerable en el sensor, partes de la escena pueden aparecer ocluidas o con sombras. Esto se debe a que, dependiendo de la escena, un objeto puede estar siendo proyectado por el emisor IR, pero no detectado por sensor IR, lo que se traduce en regiones sin información.

Propiedades y orientación de las superficies de los objetos Las propiedades reflectantes de las superficies de los objetos y su ángulo respecto al plano de la imagen pueden generar errores grandes en la medición e inclusive generar regiones sin información. Esto pasa comúnmente con materiales como metales, cerámicas o plásticos transparentes.

2.2. El robot Justina

Justina es un robot autónomo de servicio construido enteramente en el laboratorio de Bio-Robótica de la Facultad de Ingeniería de la UNAM. Cada año, desde el 2007, participa en la competencia internacional de robótica *RoboCup*.

El paradigma utilizado para la ejecución de la coordinación de las diferentes tareas en el Robot Justina está basado en el sistema *Virbot* desarrollado por Savage-Carmona et al. (1998), en el cual, se estructuran diferentes programas especializados para realizar tareas conjuntamente. En la figura 2.2 se observan los diferentes programas y la interconexión entre ellos para lograr dicho paradigma.

Como parte del programa que realiza las funciones de *Percepción* del sistema VirBot y para cumplir con las pruebas a las que está sometido durante estas competencias, el robot debe poseer la capacidad de, entre muchas otras cosas, detectar, reconocer y manipular los objetos que se encuentran en su entorno.

La capacidad de detectar y reconocer objetos se logra mediante un sensor Kinect dispuesto en el robot, el cual proporciona información de su entorno. Esta información viene dada de dos maneras: la primera es una imagen de color en el espacio RGB, es decir, una fotografía digital de la escena, y la segunda es un mapa de profundidad, el cual proporciona información sobre la distancia que existe entre el sensor y las superficies de los objetos de la escena.

El sensor Kinect en el robot Justina se utiliza para otras tareas además de la detección e identificación de objetos, como son la clasificación de espacio entre libre y ocupado al momento de la navegación, la busqueda de personas mediante la detección de sus rostros o la identificación de gestos naturales para llamar su atención. Por tal motivo, surge la necesidad de ampliar el ángulo de visión que aporta este sensor. Para esto, se colocó sobre una cabeza mecatrónica que permite moverlo y capturar una escena desde diferentes puntos de vista, sin tener que mover al robot.

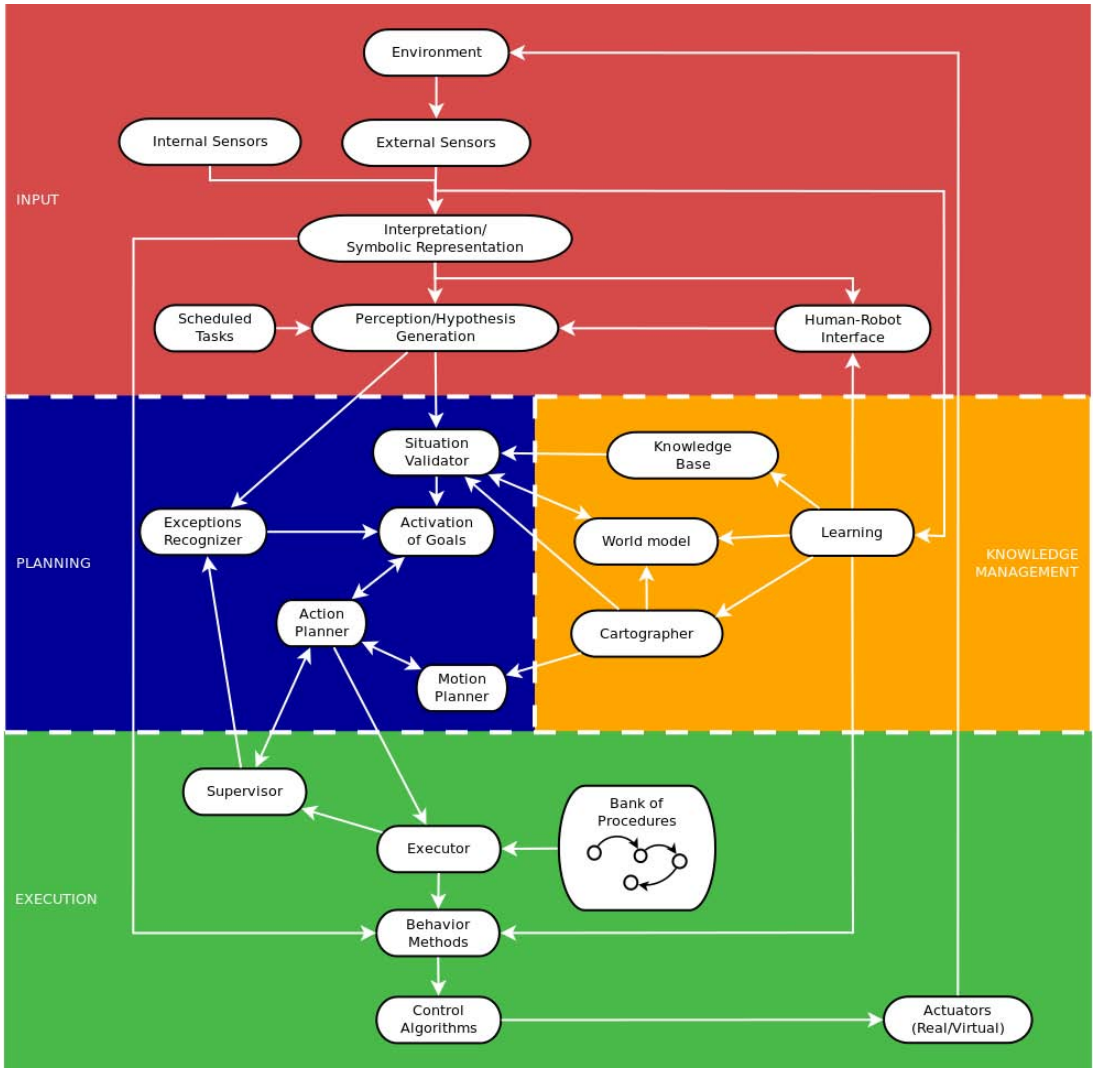


Figura 2.2: Diagrama del paradigma VirBot.

Figura 2.3: Posición del sensor Kinect en el robot Justina.

La cabeza mecatrónica está localizada en la parte más alta del robot, en el extremo de un poste que se encuentra en el centro del mismo, visto desde el frente. Está construida utilizando dos servomotores, que le permiten modificar su inclinación respecto al horizonte y al mismo tiempo girarla con respecto al frente del robot, como se muestra en la figura 2.3.

Asimismo, estos servomotores permiten conocer en todo momento los ángulos de giro θ y ϕ, los cuales corresponden a los ángulos de guiñada y cabeceo (en inglés *pan* y *tilt*), respectivamente, y tienen un radio máximo de giro de 2 radianes.

2.3. El módulo de visión

El sistema de software que maneja el hardware del robot utiliza una arquitectura *Blackboard*. Este tipo de arquitectura es una colección de programas independientes que trabajan cooperativamente entre sí, comunicados en una estructura centralizada a través de un programa denominado Blackboard. Este último es el encargado de gestionar la comunicación entre los diversos programas, además de servir como un almacén central de datos.

En nuestro sistema, cada programa especializado se denomina *Módulo*. La mayor parte de la comunicación entre módulos se realiza utilizando un protocolo de comando-respuesta a través de sóckets *TCP/IP*. En la figura 2.4 se muestra un diagrama de los módulos que componen actualmente al robot Justina.

Utilizando este protocolo, cada módulo del sistema pude recibir comandos para que realice determinada tarea. Opcionalmente, cuando se envía un comando, se le pueden añadir parámetros para la ejecución de dicha tarea, utilizando la siguiente sintaxis:

```
nombre_comando [parámetro1, parámetro2, ... , parámetroN]
```

Asimismo, cuando un módulo recibe un comando y ejecuta la tarea determinada por este, debe informar al módulo que hizo la petición sobre el estado de la tarea, mediante un 1 o un 0, dependiendiendo si la tarea se cumplió exitosamente o no, respectivamente. De igual manera, la respuesta puede contener parámetros que informen sobre los valores que se generaron al realizar dicha tarea. Entonces, la sintaxis de una respuesta es la siguiente:

```
1|0 nombre_comando [valor1, valor2, ... , valorN]
```

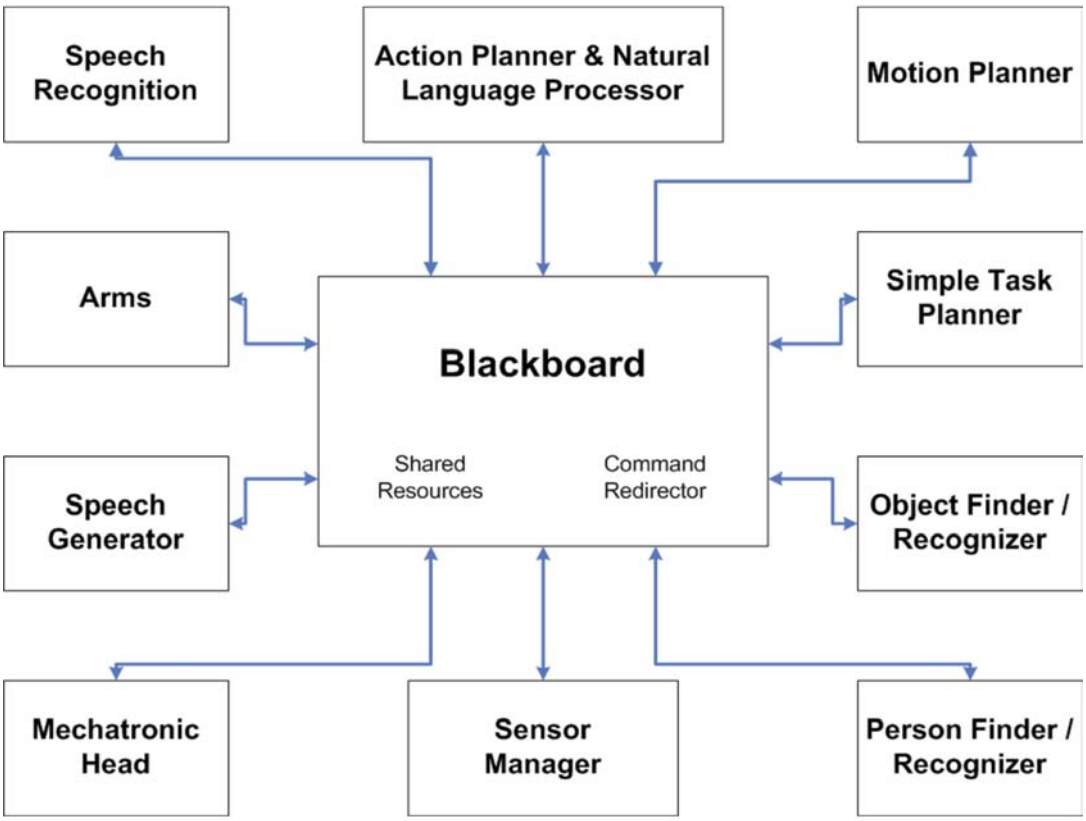


Figura 2.4: Diagrama de módulos del robot Justina.

En ese sentido, un módulo puede requerir datos adicionales o información sobre el estatus del robot para realizar un comando en específico, permitiendo a los algoritmos actuar con un mayor conocimiento del entorno. Un ejemplo de esto pueden ser los ángulos actuales de giro de la cabeza, los cuales pueden ser analizados y utilizados dentro de comandos que no pertenezcan al módulo "Cabeza", como por ejemplo, para conocer la dirección de la escena que esta siendo observada por las camaras montadas en la cabeza.

Dentro de estos módulos, el encargado de administrar la conexión con sensor Kinect y de procesar la información que genera este sensor es el módulo de Visión, el cual está desarrollado utilizando el lenguaje de programación *C++*.

Nombre	Canales	Tipo de Dato	Unidad
Mapa de profundidad	1	unsigned char de 16 bits	milímetros
Nube de puntos organizda	3	float	metros
Mapa de disparidad	1	unsigned char de 8 bits	-
Máscara de valores válidos	1	unsigend char de 8 bits	-
Imagen BGR	3	unsigned char de 8 bits	-

Tabla 2.2: Imágenes del sensor obtenidas con OpenNI.

Para gestionar la conexión con el sensor y para leer los datos del sensor se utiliza el framework OpenNI (*Open Natural Interaction*). OpenNI es un proyecto *Open Source*, el cual es famoso por ser el primer SDK no oficial en soportar el sensor Kinect (Falahati, 2013).

De igual manera, el módulo de Visión hace uso de la librería OpenCV (*Open Computer Vision*). Esta librería proporciona alrededor de 500 funciones, enfocadas a construir soluciones para problemas de procesamiento digital de imágenes y visión computacional. Se escribió en *C* muy optimizado, con el objetivo de que fuera computacionalmente muy eficiente para poder desarrollar aplicaciones en tiempo real (Bradski and Kaehler, 2008).

Utilizando OpenNI y OpenCV en conjunto, se puede acceder fácilmente a los datos generados por el sensor Kinect utilizando las estructuras que requieren los métodos de OpenCV. La información obtenida y las características de su estructura se presentan en la tabla 2.2.

Capítulo 3

Detección de objetos

3.1. Transformación de la nube de puntos

Cada pixel en la nube de puntos organizada contiene información de su posición en el espacio de 3 dimensiones. Esta posición está determinada en un sistema de referencia con origen O_0 en el centro del sensor Kinect. Los ejes X, Y y Z de este sistema de referencia apuntan hacia el frente, hacia la izquierda y hacia arriba, respectivamente [1], cuando los ángulos de la cabeza tienen valores de 0.

Dado que el sensor está montado en la cabeza mecatrónica, la cual tiene dos grados de libertad (guiñada y cabeceo) y esta se mueve para apuntar hacia donde exista información relevante del mundo, el sistema de referencia de los puntos no es estático, por lo que, como primer paso, se transforma cada punto al sistema de referencia O_2 del robot, como se puede ver en la figura 3.1.

Como se detalla más adelante, la transformación de los puntos permite una rápida y eficiente identificación de los planos horizontales, dado que con esto se consigue que el ángulo entre las normales de los planos horizontales y el eje Z sea aproximadamente 0; es decir, el eje Z apunta hacia arriba. Además, durante el reconocimiento de objetos, esto permite simplificar los cálculos para la obtención de características basadas en información 3D.

[1] Viendo el Kinect desde atrás.

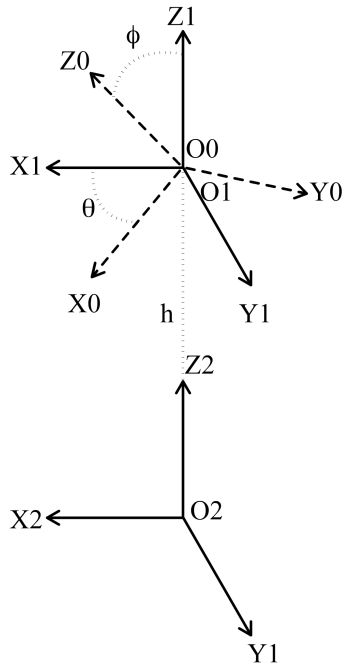


Figura 3.1: Sistemas de referencia del Kinect y del robot.

Para transformar un punto $p_{0i} = (x_{0i}, y_{0i}, z_{0i})$, donde i representa su índice en una dimensión en la nube de puntos organizada, del sistema O_0, a su correspondiente punto p_{2i}, en el sistema O_2, se utiliza la siguiente tranformación homogénea, la cual involucra dos rotaciones seguidas de una traslación:

$$p_{2i} = \begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & h \\ 0 & 0 & 0 & 1 \end{bmatrix} p_{0i} \tag{3.1}$$

donde θ y ϕ son los ángulos de movimiento de la cabeza, guiñada y cabeceo, respectivamente, y h es la altura a la que se encuentra la cabeza con respecto al centro del robot.

Al final, durante este paso, se eliminan los puntos que se consideran parte del piso, dado que se determinó que los objetos que se encuentren en el piso,

corresponden en su mayoría a muebles y no es necesario detectarlos utilizando este sistema. Esto se realiza descartando los puntos para los que, despues de ser transformados, su valor de la componente z se encuentre por debajo de cierto umbral Z_t, el cual generalmente se establece como $Z_t = 0.10$.

Al final, el valor de cada punto $p(x, y, z)$ queda establecido por el algoritmo 1.

Algoritmo 1 Transformación de la nube de puntos

Entradas

P - Nube de puntos organizada
Z_t - Umbral del piso

Función KINECT2ROBOT(P, Z_t)
  for all $p(x, y, z) \in P$ **do**
    $x \leftarrow x\cos\theta\cos\phi - y\sin\theta + z\cos\theta\sin\phi$
    $y \leftarrow x\sin\theta\cos\phi + y\cos\theta + z\sin\theta\sin\phi$
    $z \leftarrow h - x\sin\phi + z\cos\phi$
    if $z < Z_t$ **then**
      $x \leftarrow 0$
      $y \leftarrow 0$
      $z \leftarrow 0$
  return P

3.2. Extracción de los planos horizontales

Dado que se requiere que los objetos estén sobre planos horizontales, por ejemplo mesas, libreros o repisas, lo primero que se hizo es identificar todos los planos de la escena, a excepción del piso, como se mencionó anteriormente.

Esto se realizó filtrando la nube de puntos, calculando las normales locales de cada punto y después, mediante una modificación del algorimto de RANSAC, se determinó el modelo matemático de cada plano.

3.2.1. Filtrado

La nube de puntos obtenida mediante la cámara RGB-D del robot (sensor Kinect), contiene un ruido considerable proporcional a la distancia a la que se encuentran los objetos. Este ruido afecta el cálculo de normales locales dando origen a normales con orientaciones no uniformes.

Para reducir el efecto de este ruido, antes de proceder con el cálculo de normales locales, se emplea un filtro de bloque de tamaño k, donde k es un entero impar mayor a 1. Con este filtro, se reemplaza el valor de cada pixel con el promedio de su vecindad en una área de tamaño $k \times k$.

Este filtro se aplica convolucionando la matriz de la ecuación 3.2 (kernel del filtro) a la nube de puntos organizada.

$$K = \frac{1}{k^2} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix} \quad \forall k > 0, k \in \text{impares} \tag{3.2}$$

3.2.2. Cálculo de las normales locales

Para identificar planos, se localizan regiones con curvaturas similares en la nube de puntos. La curvatura en un punto p_i en un región suave se puede determinar como la normal del plano tangente al punto p_i que se adecúe mejor a la región conformada por los k vecinos más cercanos de p_i en un radio r.

Para calcular estas curvaturas, se utiliza la técnica empleada por Holz et al. (2012), la cual está diseñada para calcular eficientemente las normales locales en cada pixel de una nube de puntos organizada.

La eficiencia de esta técnica se basa en considerar a los vecinos más cercanos en el espacio de la imagen, dado que se trata de nubes de puntos organizadas, en lugar de buscarles en el espacio de $\mathcal{R}3$, es decir, utilizar a pixeles vecinos para estimar el plano que mejor se adecúe a la región de interés, como se muestra en la figura 3.2. Esto conlleva una pérdida en la exactitud de las normales estimadas,

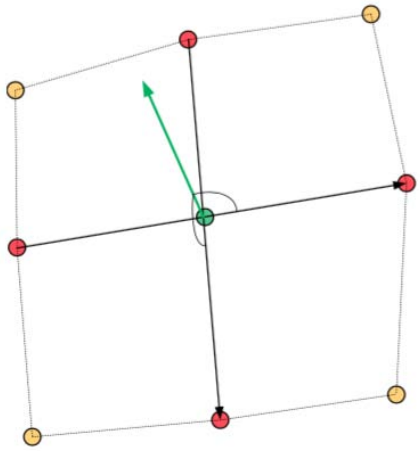


Figura 3.2: Curvatura usando los pixeles vecinos.

pero acelera su obtención, lo cual es afín con los objetivos de este trabajo y no representa un desventaja, como se verá posteriormente.

La normal n de un plano se puede obtener calculando el producto cruz de dos vectores, u y v no colineales que lo conformen, utilizando la ecuación 3.3.

$$n = u \times v \tag{3.3}$$

Para un punto cualquiera $p_{i,j} = (x, y, z)$ de la nube de puntos organizada, donde i y j corresponden a su renglón y a su columna, respectivamente, los vectores u y v utilizados para calcular su normal se obtienen del valor de los pixeles vecinos de la nube de puntos, como se muestra en la figura 3.3, usando las siguientes ecuaciones:

$$u = p_{i,j+1} - p_{i,j-1} \tag{3.4}$$

$$v = p_{i-1,j} - p_{i+1,j} \tag{3.5}$$

Una vez obtenidos los vectores del plano correspondientes para un punto, se utiliza la ecuación 3.3 para obtener la normal en cada punto. De las normales

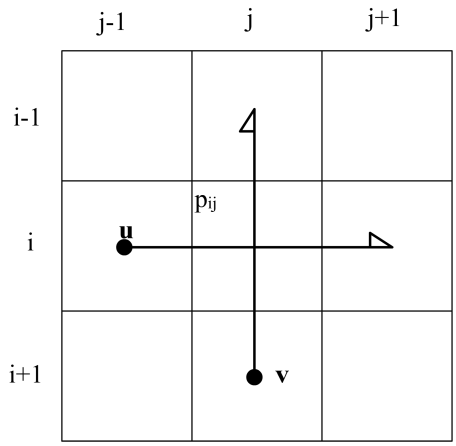


Figura 3.3: Vectores para el cáclulo de las normales locales.

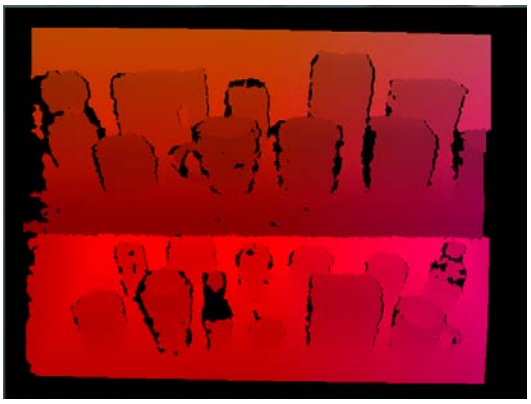
calculadas, se obtiene su vector unitario, dado que se requiere en pasos posteriores para simplificar cálculos.

Ademas, las normales que se encuentren por debajo del plano XY, es decir, aquellas cuya componente en z sea menor a cero, se multiplican por -1. Esto ayuda a reconocer con mayor exactitud los planos, dado que no es relevante para el algoritmo de detección si las normales cercanas a la horizontales apuntan hacia arriba o hacia abajo.

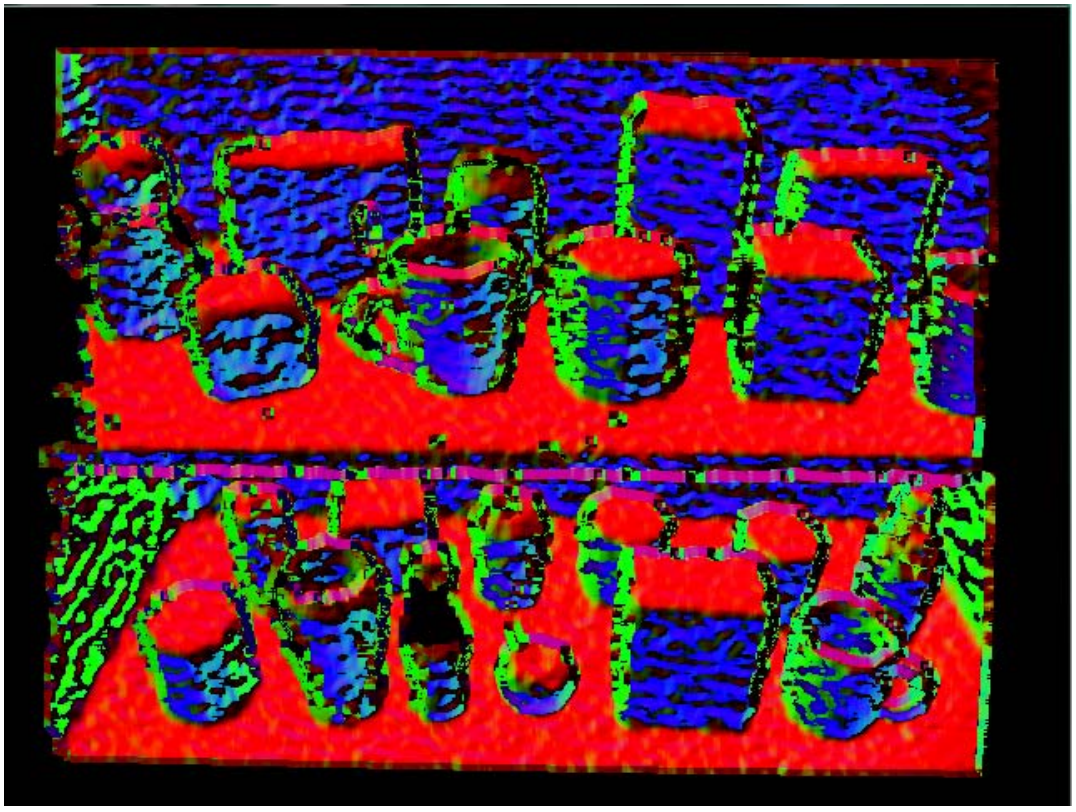
Los pasos de los cálculos para la obtención de las normales locales se resumen en el algoritmo 2 y un ejemplo del resultado de este algoritmo se puede observar en la figura 3.4.

Como paso siguiente, utilizando la imagen de las normales locales, se crea una máscara binaria con la información de las normales que presentan un valor de su componente en z por arriba de un umbral (por ejemplo, $z > 0.9$), la cual servirá para identificar los puntos en los que las normales apuntan hacia arriba y poder identificar los planos horizontales.

(a) Imagen RGB.



(b) Nube de puntos organizada.



(c) Normales locales.

Figura 3.4: Resultado del cálculo de las normales locales para un nube de puntos organizada.

Algoritmo 2 Cálculo de las normales locales.

Entradas

P - Nube de puntos organizada de tamaño $M \times N$

z_{thres} - Umbral para considerar una normal vertical

Salida

N - Normales organizadas de tamaño $M \times N$

V - Máscara binaria con normales verticales de tamaño $M \times N$

1: **Función** GETNORMALS(P)
2: $N(i,j) = (0,0,0)\ \forall\ 0 < i < M, 0 < j < N$
3: $V(i,j) = (0)\ \forall\ 0 < i < M, 0 < j < N$
4: **for** $i = 1$ to $M - 1$ **do**
5: **for** $j = 1$ to $N - 1$ **do**
6: $\mathbf{u} \leftarrow p_{i,j+1} - p_{i,j-1}$
7: $\mathbf{v} \leftarrow p_{i-1,j} - p_{i+1,j}$
8: $\mathbf{n} \leftarrow \mathbf{u} \times \mathbf{v}$
9: **if** $z_n < 0$ **then**
10: $\mathbf{n} \leftarrow -\mathbf{n}$
11: **if** $|n| \neq 0$ **then**
12: $n \leftarrow \frac{n}{|n|}$
13: $N(i,j) \leftarrow n$
14: **if** $z_n > z_{thres}$ **then**
15: $V(i,j) \leftarrow 1$
16: **return** N

3.2.3. Identificación de los planos

Para identificar los planos existentes en una escena, se utiliza el algoritmo de RANSAC sobre los puntos de la nube de puntos que tengan normales verticales. Esto, junto con el método de PCA para refinar el modelo de los planos, permiten una identifación robusta.

El algoritmo RANSAC (*RANdom SAmple Consensus*) fué publicado por Fischler and Bolles (1981). Es un método iterativo utilizado para adecuar un modelo a un conjunto de datos experimentales, el cual es capaz de interpretar y suavizar este conjunto aunque contenga un porcentaje significativo de datos erróneos o valores atípicos (denominados comúnmente *outliers*); es decir, que no se adecúan al

mejor modelo. Esto lo hace ideal para problemas de análisis automático de imágenes, en donde la interpretación está basada en los datos obtenidos mediante detectores de características propensos a errores.

A diferencia de otros métodos de estimación de parámetros y de adecuación de modelos, los cuales trabajan usando la mayor cantidad de datos posible para estimar un modelo inicial y despúes tratan de eliminar los *outliers*, el primer paso de RANSAC es obtener un conjunto con el número mínimo de datos necesarios para estimar un modelo, para posteriormente añadir a este conjunto los datos que más se adecúen a la propuesta inicial. Si el número de datos en el nuevo conjunto es congruente con la solución buscada, los parámetros del modelo se refinan utilizando este nuevo conjunto y se obtiene una solución. Por el contrario, si el número de datos es menor al esperado, se selecciona un nuevo conjunto de datos mínimo y se repite el proceso hasta encontrar una solución o hasta que se cumpla con una condición de paro, comúnmente, un número máximo de iteraciones.

Una de las desventajas de utilizar este método es que, si el número de *outliers* es mayor al 50 % de los datos, el desempeño puede no ser muy bueno (Hast et al., 2013), por lo que al algoritmo clásico se le pueden hacer dos modificaciones para inceremen tar su eficiencia:

- Utilizar una heurística en lugar de un proceso aleatorio para seleccionar los puntos que generan el primer estimado de los parámetros del modelo. Con esto se puede acelerar la convergencia de este método.

- Utilizar el conjunto de datos obtenidos al final, para reestimar y afinar un nuevo modelo que se ajuste a más datos, tantas veces como sea necesario, con el propósito de obtener un modelo con mayor exactitud.

Para estimar el modelo de un plano, el número mínimo de puntos necesarios no colineales son 3, por lo que, como primer paso para el algoritmo implementado, se escogen aleatoriamente los puntos p_1, p_2 y p_3, de la nube de puntos, para los cuales sus normales sean verticales. Con estos puntos, se obtienen dos vectores v_1 y v_2 con los que, el producto cruz de estos será la normal del plano n y cualquiera de estos un punto de este.

$$v_1 = p_2 - p_1$$
$$v_2 = p_3 - p_1$$
$$n = \frac{v_1 \times v_2}{|v_1 \times v_2|}$$

Usando n y el el punto p_1, los parámetros del plano $\pi : Ax+By+Cz+D = 0$ quedan como:

$$A = n_x$$
$$B = n_y$$
$$C = n_z$$
$$D = -(n_x p_{1x} + n_y p_{1y} + n_z p_{1z})$$

Aquí, se usaron dos heurísticas para acelerar la convergencia del método:

- Se espera obtener planos de un tamaño considerable, por lo que los puntos elegidos para el modelo deben de estar alejados una cierta distancia d, es decir:
$$d > |p_1 - p_2|, |p_1 - p_3|, |p_2 - p_3| \tag{3.6}$$

- Se desea buscar solamente planos horizontales, por lo que se verifica que el valor de la componente de n_z sea mayor a un cierto umbral z_u, es decir:
$$n_z > z_u \tag{3.7}$$

Aunque la elección de los puntos para el modelo inicial siguen siendo aleatorios, con estas heurísticas se evita que el método entre en el proceso de recorrer todos los puntos para validar su adecuación al modelo.

Una vez se tiene un modelo candidato de plano, se verifica para cada punto, cuya normal sea vertical, la distancia a este. Si esta distancia es menor a un cierto umbral d_π, se dice que el punto pertence al plano y se almacena en una lista. Para cualquier punto p_i, este se almacena si cumple con la siguiente ecuación:

$$d_\pi > \frac{|Ap_{ix} + Bp_{iy} + Cp_{iz} + D|}{\sqrt{A^2 + B^2 + C^2}} \tag{3.8}$$

Si el número de puntos almacenados es mayor a un valor predeterminado $N_{inliers}$, se considera que se encontró un plano en la escena. Este valor predeterminado se obtiene experimentalmente y se basa en el tamaño y distancia de los planos. Además, estos puntos son removidos de la lista original para continuar buscando más planos dentro de la misma escena.

Para obtener un modelo refinado del plano, considerando todos los puntos agregados a la lista, se utiliza la técnica de PCA, la cual ya ha sido utilizada en robótica para estimar planos de un conjunto de puntos, por ejemplo por Weingarten et al. (2004).

La técnica de PCA (*Principal Component Analysis*), desarrollada por Pearson (1901), es un método estadístico que consiste en encontrar una transformación ortogonal para un conjunto de datos de dimension d a una base donde los ejes coordenados corresponden con vectores ortogonales en los que, en orden decreciente, apuntan en la misma dirección que la varianza de los datos. Con esto se consigue que los puntos transformados, denominados componentes principales, sean combinaciones lineales no correlacionadas de los datos originales.

Una forma de obtener los nuevos ejes para PCA consiste en calcular los eigenvectores $\{v_1, \ldots, v_d\}$ de la matriz de covarianza de los datos . El valor de los eigenvalores $\{\lambda_1, \ldots, \lambda_d\}$ correspondiente a cada eigenvector, determinará un estimado de la varianza de los datos en ese eje: entre mayor sea, más varianza existirá.

Uno de los principales usos de esta técnica es el de reducir la dimensionalidad de un conjunto de vectores, los cuales pueden estar altamente correlacionados y, por tanto, tendrían información redundante. Identificando durante este análisis

a los componetes que representan a la mayoría de los datos, se puede prescindir de los componentes que presenten información correlacionada, es decir, información redundante, para así trabajar con la misma cantidad de información pero utilizando un menor número de datos.

Aplicando PCA, el plano que mejor se adecúa a una nube de puntos en 3D es aquel cuya normal n corresponde al eigenvector v_i con el valor propio λ_i más pequeño, lo cual significa que la varianza sobre este vector es mínima. Asimismo, el punto $p = (x, y, z)$ para determinar la ecuación del plano será el promedio de los puntos, lo que se conoce como su centro de gravedad, como se muestra a continuación:

$$p = \frac{\sum_{i=1}^{n} p_i}{n} \tag{3.9}$$

Por lo que los párametros del plano refinado, utilizando el eigenvector v_i cuyo eigenvalor λ_i es el menor, con todos los puntos cercanos como entrada, quedan como:

$$\begin{aligned} A &= v_{ix} \\ B &= v_{iy} \\ C &= v_{iz} \\ D &= -(v_{ix}p_x + v_y p_y + v_z p_z) \end{aligned}$$

Una vez obtenido el modelo de un plano refinado, se repite nuevamente el proceso de verificar la distancia de cada punto con normal vertical a este modelo utilizando la ecuación 3.8, agregar a una lista los puntos cercanos y obtener el modelo final utilizando PCA. Experimentalmente se comprobó que esto agrega un número considerable de puntos al modelo por lo que la estimación es mejor. Además, estos puntos son removidos de la nube de puntos para continuar con la búsqueda de nuevos planos.

La implementación del algoritmo de RANSAC, adecuado para la extracción de planos como se detalló anteriormente, se encuentra resumida en el algoritmo 3.

Algoritmo 3 Implementación de RANSAC para la búsqueda de planos.

Entradas

P - Nube de puntos organizada de tamaño $M \times N$.
Q - Normales organizadas de tamaño $M \times N$.
i_{max} - Número máximo de iteraciones.
p_{min} - Número mínimo de puntos para considerar un plano.
d_{min} - Distancia mínima entre un punto y el plano para considerarlo.
z_{min} - Valor mínimo para considerar una normal como vertical.

Salida

ListaPlanos - Lista de ecuaciones de los planos encontrados

Función GetPlanesRANSAC(P).
  $i \leftarrow 0$
  while $i++ < i_{max}$ **do**
    $p_1 \leftarrow$ Punto aleatorio de P con normal vertical
    $p_2 \leftarrow$ Punto aleatorio de P con normal vertical
    $p_3 \leftarrow$ Punto aleatorio de P con normal vertical
    if $d < |p_1 - p_2|$ or $d < |p_1 - p_3|$ or $d < |p_2 - p_3|$ **then**
      continue
    $\pi(A, B, C, D) \leftarrow ObtenerPlano(p_1, p_2, p_3)$
    if $n_z < z_{min}$ **then**
      continue
    for all $p_i \in P$ **do**
      $d = (|Ap_{ix} + Bp_{iy} + Cp_{iz} + D|)/(\sqrt{A^2 + B^2 + C^2})$
      if $d_{min} > d$ **then**
        $Inliers.Agregar(p_i)$
    if $Inliers.Size < p_{min}$ **then**
      continue
    $\pi(A, B, C, D) \leftarrow PCA(Inliers)$
    $Inliers.clear()$
    for all $p_i \in P$ **do**
      $d = (|Ap_{ix} + Bp_{iy} + Cp_{iz} + D|)/(\sqrt{A^2 + B^2 + C^2})$
      if $d_{min} > d$ **then**
        $Inliers.Agregar(p_i)$
    $\pi(A, B, C, D) \leftarrow PCA(Inliers)$
    $P.Remover(Inliers)$
    $ListaPlanos.Agregar(\pi)$
  return *ListaPlanos*

Asimismo, un ejemplo de los planos obtenidos utilizando este método se puede observar en la figura 3.5.

3.3. Segmentación de los objetos

Para cada modelo de los planos obtenidos como se describió en la sección anterior, se crea una máscara $V(i, j)$ que indica los pixeles correspondientes a objetos, utilizando la distancia d de los puntos a los planos: si esta se encuenta en entre un valor mínimo $d_{min} > 0$ y un máximo d_{max} acorde al tamaño de los objetos que se desean encontrar, como se muestra a continuación:

Una vez obtenida la máscara con valores binarios que corresponden a los objetos que se encuentran sobre los planos identificados, es necesario segmentar cada uno de dichos objetos, es decir, asignar cada pixel de la máscara a un objeto en particular, para así separarlo de los démas y pasarlo a la siguiente fase del proceso el cual es la identificación de los objetos.

Para esto, se parte de la idea de que los puntos que pertencen a un mismo objeto, serán necesariamete pixeles vecinos en la nube de puntos organizada, aunque esto no suceda al contrario, es decir, pixeles vecinos de la mascara no necesariamente pertencen a un mismo objeto. Esto no es necesariamente cierto para todos los casos, como por ejemplo, si la parte media de un objeto está obstruida por otro y sólo se visualizan sus extremos. Estos casos no son considerados en el presente trabajo, pero se propone como un problema a resolver como trabajo futuro.

En este sentido, se asigna a cada pixel $p_i(x_i, y_i) \neq 0$ de la máscara de objetos, una etiqueta de un conjunto $L = (L_1, L_2, \ldots, Ln)$ donde n es el número de objetos segmentados. Para esto se considera lo siguiente :

1. Todos los $p_i(x_i, y_i) \neq 0$ de la máscara binaria deben pertenecer a una etiqueta de L, es decir si P es el conjunto de todos los puntos $p_i(x_i, y_i) \neq 0$ y L es el cunjunto de etiquetas asignadas a cada punto, $L_1 \cup L_2 \cup \cdots \cup L_n = P$

(a) Planos obtenidos.

(b) Planos en la la imagen RGB.

Figura 3.5: Planos detectados usando el algoritmo propuesto.

2. Cada punto $p_i(x_i, y_i) \neq 0$ debe pertenecer sólamente a una etiqueta, es decir, si L_j y L_k son un conjunto de puntos asignados a cada etiqueta, entonces $L_j \cap L_k = \emptyset$, $\forall j \neq k$

Para verificar si dos pixeles vecinos pertencen a un mismo objeto, se utiliza lo que se concoce como predicado, es decir, una función que permita saber si dos vecinos pertenecen a la misma etiqueta. Para el predicado de esta implementación, se utiliza la distancia euclidiana entre pixeles vecinos de la nube de puntos organizada y se define una distancia de umbral d_{min}, obtenida experimentalmente. Si la distancia euclidiana entre dos vecinos es menor que d_{min} se dice que estos pertenen al mismo objeto. Esto es, si p_1 es un vecino de p_2 en la nube de puntos organizada y si $|p_1 - p_2| < d_{min}$, entones $p_1 \in L_i$ y $p_2 \in L_i$.

Una desventaja de este algoritmo es que, si dos objetos se encuentran a una distancia menor que d_{min} estos serán segmentados como un mismo objeto. Afortunadamente, acorde a las reglas de la competencia, los objetos siempre estarán a una distancia considerable uno del otro.

En esta implementación se utiliza la conectividad-8 entre pixeles, es decir, para un pixel p con coordenadas (x, y), se tiene que sus vecinos con conectividad 8 $V_8(p)$ son los pixeles con coordenadas: $V_8(p) = \{(x+1, y), (x+1, y), (x, y+1), (x, y-1), (x+1, y+1)$, a $(x-1, y-1), (x+1, y-1), (x-1, y+1)\}$

Para este algoritmo, sólo se requiere recorrer una vez todos los puntos de la imagen. Se utiliza la nube de puntos organizada, y la máscara de objetos. Entonces, se recorre la nube de puntos, si se encuentra un pixel que no ha sido etiquetado en la máscara, se crea una nueva etiqueta, se le asigna a este punto y se ingresa a una estructura de datos tipo cola. Mientras que la cola no esté vacía, se obtienen pixeles de esta y se verifican sus vecinos. A cada vecino que no esté todavía etiquetado y que cumpla con el predicado, se le asigna la misma etiqueta y se ingreasa a la cola. De esta forma, se etiquetan uno por uno los pixeles de la máscara de objetos, verificando sistemáticamente sus vecinos. El método descrito anteriormente se puede observar en el algortimo 4.

Un ejemplo de la aplicación del algoritmo 4 se observa en la figura 3.6.

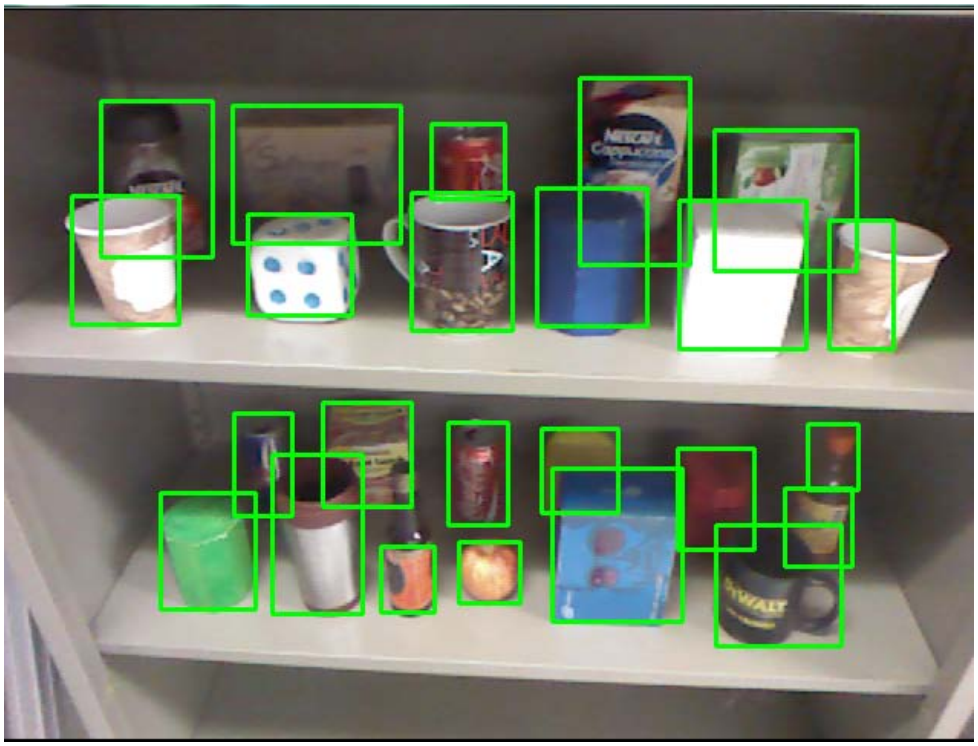
Algoritmo 4 Segmentación 3D de objetos.

Entradas

P - Nube de puntos de los objetos de tamaño $M \times N$.
V - Máscara binaria de los objetos en el plano de tamaño $M \times N$.
d_{min} Distancia mínima entre los pixeles vecinos.

```
1:  Función ConnectedComponets3D(P, V, minDist)
2:      L(i, j) = 0 ∀ 0 < i < M, 0 < j < N        ▷En L se guardan las etiquetas.
3:      l ← 0
4:      for all v = 1 ∈ V do
5:          if L(v) ≠ 0 then
6:              continuar
7:          l ← l + 1
8:          L(v) ← l
9:          Cola.Agregar(v)
10:         while ColaVecinos.Size() > 0 do
11:             p ← Cola.Sacar()
12:             for all vecino q de p do                ▷Para cada pixel vecino
13:                 if V(q) = 1 and |p − q| < d_min then
14:                     L(q) ← l
15:                     Cola.Agregar(q)
16:     return L
```

(a) Objetos y sus respectivas etiquetas.



(b) Objetos detectados en la imagen RGB.

Figura 3.6: Objetos detectados y segmentados.

Capítulo 4

Reconocimiento de objetos

4.1. Descripción basada en información 3D

Resultado del proceso de segmentación de los objetos sobre planos horizontales descrito anteriormente, se cuenta con los índices de los puntos de la nube de puntos que conforman cada objeto, además de la ecuación del plano en el que se encuentra dicho objeto.

Dada esta información, se diseñó un método que proporciona el objeto más parecido entre uno entrenado previamente y uno detectado en la escena basado en dos características simples pero importantes que al combinarlas describen de manera robusta a cualquier objeto tridimensional: la forma y el tamaño.

4.1.1. Error de tamaño

Una de las características más básicas de un objeto es su tamaño. Dada la nube de puntos de los objetos, se puede extraer esta propiedad de cada uno de estos y utilizarla para discriminar de manera muy robusta candidatos para un objeto entrenado.

En el presente trabajo se afirma que el tamaño es una característica robusta porque, en muchos casos, incluyendo a los de competencia, es difícil encontrar objetos que tengan las mismas medidas en las tres dimensiones que se mane-

jan comúnmente, es decir, la altura, la anchura y la profundidad, si estos no pertenecen a la misma clase de objetos.

La resolución y ruido del sensor Kinect dificultan la extracción de puntos característicos y descriptores basados en 3D. Sin embargo, en distancias cortas (que es como comúnmnete se utiliza para hacer reconocimiento de objetos) el error de profundidad es de aproximadamente 2 mm a 1 m de distancia (Khoshelham, 2011), por lo que se puede decir que es pequeño comparado con el tamaño de los objetos. Esto permite tener una aproximación precisa de las dimensiones medidas, para los objetivos descritos en el presente trabajo.

Para obtener una medida aproximada del tamaño de un objeto, se encuentra lo que se conoce como caja delimitadora orientada (*OBB, oriented bounding box*). Una caja delimitadora es un volumen cerrado de 6 caras que contiene completamente la unión de todas las partes de un objeto. Esta caja, al ser orientada, se caracteriza por no estar restringida a que sus tres ejes estén alineados a un marco de referencia global.

Una buena forma de calcular el volumen de un objeto es con lo que se denomina caja orientada delimitadora mínima, sin embargo, los métodos para obtener la caja delimitador orientada minínima tienen un complejidad, en el peor de los casos, de $\mathcal{O}(n^3 \log n)$ (O'Rourke, 1985), por lo que podemos considerarlos algoritmos costos, computacionalmente hablando.

Para aumentar la velocidad del algoritmo, se decidió hacer uso de una técnica con la cual, no se garantiza la obtención de la caja delimitiadora orientada mínima, pero se aproxima mucho a esta, considerando que los objetos que se deben reconocer durante las pruebas en competencias se basan en figuras simétricas, utilizando como plano de simetría un plano paralelo al plano en el que está el objeto, por ejemplo, cajas o cilindros.

Para obtener las medidas que corresponden al ancho y profundidad de un objeto, lo que se hace es proyectar todos los puntos de la nube de puntos del objeto en el plano en el que se encuentra el objeto, siguiendo la dirección del vector normal de dicho plano. Una vez proyectados los puntos en el plano, estos se pueden tratar como puntos en 2D y obtener el rectángulo delimitador mínimo,

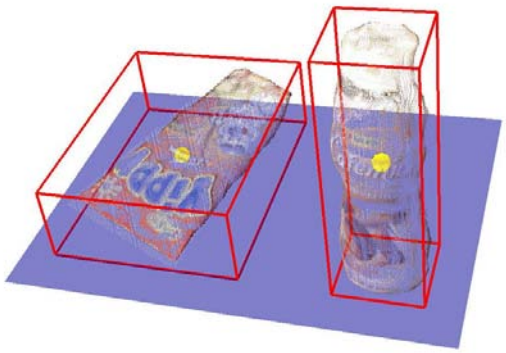


Figura 4.1: Caja delimitadora no orientada para diferentes objetos.

utilizando el algoritmo propuesto por Freeman and Shapira (1975), el cual viene implementado en la librería de OpenCV y nos entrega las dimensiones, ancho y profundidadad de un rectángulo, además de su orientación en el plano.

Para obtener un volumen a partir del rectángulo de área mínima, es necesario contar con la altura que tendrá dicho rectángulo. Para esto simplemente se iteró sobre todos los puntos de la nube de puntos, obteniendo la distancia del punto al plano donde se encuentra el objeto y se guarda la distancia mayor, es decir, si $P = \{p^1, \ldots, p^n\}$, es el conjunto de todos los puntos 3D de un objeto y $\Pi : ax + by + cz + d = 0$ es el plano en el que se encuentra dicho objeto, la altura h del objeto estará dada por:

$$h = \max_{i=1\ldots n} \left(\frac{|ap_x^i + bp_y^i + cp_z^i + d|}{\sqrt{a^2 + b^2 + c^2}} \right) \tag{4.1}$$

Con esto, se logra obtener la terna $V = \{w, d, h\}$, la cual servirá para tener un cálculo del tamaño del objeto. Las cajas delimitadoras obtenidas a partir deb este método son similares a las que se pueden observar en la figura 4.1.

De este modo, obteniendo las cajas delimitadoras de los objetos entrenados y las de los objetos de la escena, se puede calcular rápidamente lo que se denominó el error de tamaño $e_{tamaño}$, restando cada compoentene de las dimensiones. Esto es, si V_T corresponde a las dimensiones de un objeto entrenado cualquiera, y V_i a las de un objeto en la escena, el error de tamaño estará dado por:

$$e_{tamaño} = |V_T - V_i| \tag{4.2}$$

Cuando los objetos que se quieren reconocer no cumplen con la característica de simetría que se describió anteriormente, se puede utilizar simplemente la altura como un parámetro para la estimación del tamaño de un objeto. En este caso, el error de tamaño queda simplemente como:

$$e_{tamaño} = |h_T - h_i| \tag{4.3}$$

4.1.2. Error de forma

El método propuesto para caracterizar la forma de un objeto se basa en la descripción del contorno de la figura que se obtiene de la proyección de sus puntos en el plano que lo sustenta. En un principio, esta idea surgió como una forma de identificar objetos cilíndricos, para los cuales, la figura que formaría dicha proyección sería un círculo, el cual puede ser rápidamente identificable por métodos como la transformada de Hough.

Sin embargo, al aplicar técnicas similares a las utilizadas en el reconcimiento de patrones en 2D sobre el contorno, se pueden diferenciar objetos 3D con formas regulares, con una invariancia en la escala y en la rotación sobre el eje de la normal al plano.

Para obtener la proyección de los puntos en 2D, se utilizó la simplificación que proporciona el haber transformado los puntos a las coordenadas del robot y, considerando que los planos extraídos son horizontales, simplemente se elimina la componente z de los puntos de la nube de puntos.

A continuación, se obtiene la envolvente convexa de los puntos en 2D utilizando la técnica propuesta por Graham and Yao (1983). Esta envolvente es el perímetro de un polígono regular formado por los puntos externos de los puntos en 2D. Un ejemplo del polígono generado para diferentes objetos utilizando este método se puede visualizar en la figura 4.2.

(a) Portafotos (detrás). (b) Caja. (c) Taza.

Figura 4.2: Ejemplo de polígonos regulares obtenidos de objetos 3D.

El contorno del polígono que se forma se utilizará para describir la forma del objeto tridimensional utilizando los *Momentos Invariantes de Hu*. Estos fueron introducidos como una técnica en el área de Reconocimiento de Patrones en 1962, resultado de la aplicación de la teoria de invariantes algebraicos y han sido utilizados en numerosas aplicaciónes desde entonces.

Los Momentos de Hu son 7 cantidades obtenidas que describen una imagen en 2D. Son invarianes a traslaciones, rotaciones y escala. Estos fueron utilizados por el autor para reconcer letras y números impresos Hu (1962). Además, Flusser and Suk (2006) los utilizó para reconocer objetos de formas similares a los que se obtienen con la proyección de los puntos en 2D.

Para encontrar los 7 momentos invariantes de Hu H, para una imagen binaria $I(x, y)$, se utilizan las siguientes ecuaciones:

$$
\begin{aligned}
H[1] &= \eta_{20} + \eta_{02} \\
H[2] &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
H[3] &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - 3\eta_{03})^2 \\
H[4] &= (\eta_{30} + \eta_{21})^2 + (\eta_{21} + \eta_{03})^2 \\
H[5] &= (\eta_{30} + 3\eta_{12})(\eta_{30} + \eta 12)((\eta_{30} + \eta_{21}^2 - 3(\eta_{21} + \eta_{03}^2) \\
&\quad + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})(3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2) \\
H[6] &= (\eta_{20} - \eta_{02})((\eta_{30} - \eta_{12})^2 - (\eta_{21} - \eta_{03})^2) + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} - \eta_{03}) \\
H[7] &= (3\eta_{21} + \eta_{03})(\eta_{30} + \eta_{12})((\eta_{30} + 3\eta_{12})^2 - 3(\eta_{21} + \eta_{03}^2)) \\
&\quad - (\eta_{30} + 3\eta_{12}^2)(\eta_{21} + \eta_{03})(3(\eta_{31} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2)
\end{aligned}
$$

Donde cada componente η_{pq} se refiere a uno de los momentos centralizados normalizados de orden $p + q$ de una imagen y es obtenido como:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} \quad \text{donde} \quad \gamma = \frac{p+q}{2} + 1 \quad \forall (p+q) \geq 2 \tag{4.4}$$

Como se puede ver en la ecuación anterior, los momentos centralizados normalizados se obtienen de dividir cada momento central de la imagen entre el momento central de orden 0 elevado a una potencia γ. Estos momentos centrales se obtienen con la ecuación siguiente:

$$\mu_{pq} = \sum_{x} \sum_{y} (x - x_c)^p (y - y_c)^q I(x, y) \tag{4.5}$$

Donde $I(x, y)$ es una imagen binaria por lo que sólo puede tomar valores de 1 y 0, x_c y y_c hacen referencia a las componentes del centroide de la imagen binaria obtenidas como:

$$x_c = \frac{\sum_x \sum_y xI(x,y)}{\sum_x \sum_y I(x,y)}$$
$$y_c = \frac{\sum_x \sum_y yI(x,y)}{\sum_x \sum_y I(x,y)}$$

Afortunadamente, la implementación de la extracción de los 7 Momentos Invariantes de Hu, existe como una función para una imagen 2D binaria en OpenCV , por lo que, para este trabajo se hace uso de ella.

El vector H con los Momentos de Hu se guarda durante el proceso de entrenamiento en la base de objetos para ser comparados con los Momentos de Hu de los objetos detectados y determinar los mejores candidatos.

Para comparar los momentos de Hu H_1 y H_2 de dos imágenes, los cuales generalmente son valores muy pequeños (Mehtre et al., 1997), y obtener el error de forma e_{forma}, se utiliza la siguiente ecuación:

$$e_{forma} = \sum_{i=1}^{7} \left| \frac{1}{h_1[i]} - \frac{1}{h_2[i]} \right| \tag{4.6}$$

donde

$$h_1[i] = sign(H_1[i]) \log H_1[i]$$
$$h_2[i] = sign(H_2[i]) \log H_2[i]$$

4.2. Descripción basada en color

En cada captura, la nube de puntos y la imagen RGB que proporciona el Kinect se encuentran registradas. Esto quiere decir que existe una correspondencia entre los índices de los pixeles de ambas estructuras de datos. Por tanto, utilizando los mismos índices de los puntos 3D de los objetos segmentados en pasos anteriores, se puede acceder a la información de color de los mismos.

La información que otorga el sensor Kinect sobre el color de los objetos se encuentra en el espacio de color RGB, es decir, nos proporciona una imagen de 3 canales con valores en un rango de $[0, 255]$ en donde cada canal representa los colores rojo, verde y azul, respectivamente.

La técnica aquí propuesta se basa en la extracción del histrograma de color de un objeto detectado y la comparación de los histogramas de los objetos entrenados para encontrar similitudes entre estos y elegir al mejor candidato.

4.2.1. Transformación del espacio de color

El espacio de color RGB, aunque es el más utilizado para la representación de imágenes digitales, no es ideal para los objetivos del presente trabajo. Gevers and Smeulders (1999) concluyeron, teórica y experimentalmente, que usar RGB para el reconocimiento de objetos es apropiado solamente cuando las condiciones de luz son controladas. Sin embargo, tambien concluyó que la componente del Tono (H, del inglés *Hue*), se comporta mejor ante condiciones no controladas y además es invariante a reflejos.

El Tono está presente en el espacio de color HSV, por lo que se decidió trabajar con este para realizar el reconocimiento de objetos. Las 3 componentes de este espacio de color son:

- Tono (*Hue*) .- Representa el color propiamente dicho como una combinación de los colores primarios de la luz, es decir el rojo, el verde y el azul.

- Saturación (*Saturation*) .- La saturación, también llamada pureza, describe el porcentaje de intensidad del color definido por el Tono. Una saturación máxima se reflejará en colores muy "vivos". Asimismo, entre menor sea la saturación, el tono presentará una decolaración, pasando por grises claros hasta llegar al blanco, cuando la saturación es 0, sin importar el valor de su tonalidad.

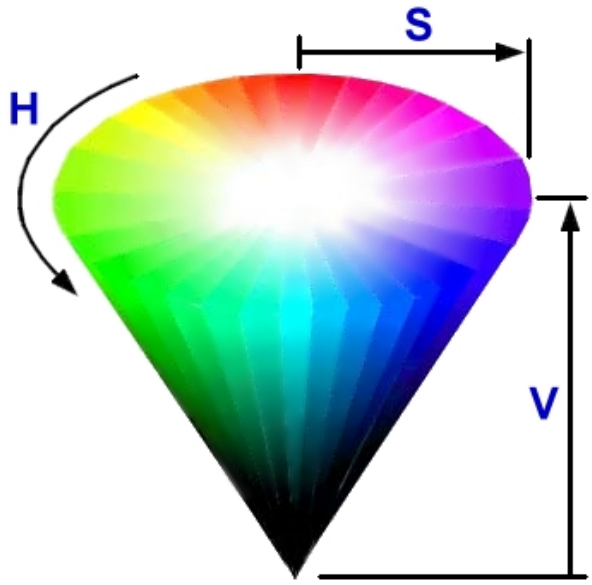


Figura 4.3: Espacio de color HSV.

- Valor (*Value*) También definido como brillo, trata de describir qué tanta luz refleja un color. Mientras esta componente se reduce, los colores tienden a grises oscuros, terminado en negro, para todos los tonos, cuando vale cero.

Es común representar el espacio HSV como un cono en el cual, el tono está representado por un valor de ángulo alrededor de la base cirular; la saturación representa la distancia desde el centro de la base al perímetro de esta y el valor por la altura del cono, como se observa en la figura 4.3.

En esta implementación, para transformar cada pixel de 3 canales del espacio de color RGB al espacio HSV se utilizaron las siguientes ecuaciones:

$$V = \text{máx}(R, G.B)$$

$$S = \begin{cases} (V - \text{mín}(R, G, B))/V & \text{si } V \neq 0 \\ 0 & \text{si } V = 0 \end{cases}$$

$$H = \begin{cases} 60(G - B)/(V - \text{mín}(R, G, B) & \text{si } V = R \\ 120 + 60(B - R)/(V - \text{mín}(R, G, B) & \text{si } V = G \\ 240 + 60(R - G)/(V - \text{mín}(R, G, B) & \text{si } V = B \end{cases}$$

Los rangos de estos valores son:

$$0 \leq H \leq 360$$
$$0 \leq V \leq 1.0$$
$$0 \leq S \leq 1.0$$

Por lo que estos se escalan para poder ocupar el tamaño de un byte, es decir, $[0, 255]$ y poder trabajar utilizando la misma estructura de imagen dentro del módulo desarrollado.

4.2.2. Obtención del histograma

Un histograma es una representación de la distribución de la frecuencia de los valores de un conjunto de datos. Para construir este, el rango de valores que pueden tomar los datos de subdivide en N_B números de clases B_i de longitud constante. Se cuenta cuántos datos se encuentran dentro del intervalo de cada clase y se registra. La unión del conteo de cada intervalo será el histograma. Por tanto, el rango de valores queda dividido como:

$$[v_{min}, v_{max}] = B_1 \cup B_2 \cup ... \cup B_N \tag{4.7}$$

Para describir un objeto utilizando su información de color, se utilizó el patrón que se genera de su histograma en el canal del Tono, en el espacio HSV. En la práctica, los colores grises muy claros y blancos, es decir, tonos con bajos niveles de Satutación, presentan una incertidumbre en cuanto a su Tono, ya que estos tienden a reflejar el color de los objetos de su alrededor así como las condiciones de luz. Lo mismo pasa con los colores para los que su Valor es pequeño, es decir, los negros y grises oscuros. Por estas razones, para el Histrogram del Tono H_H, sólo se toman pixeles con valores de Saturación y Valor por encima de 20 %. Al hacer esto, se registran los colores que lo conforman, independientemente de la iluminación a la que están sujetos los objetos.

Sin embargo, en la vida real, los objetos sí contienen blancos, negros y tonalidades de grises, por lo que al Histograma H_H, que se extrae de los objetos, se le adicionaron dos clases más: pixels con Saturación menor al 20 % H_S y pixeles con Valor menor de 20 %. Por tanto, el Histograma generado queda como:

$$\begin{aligned}
H &= H_H \cup H_S \cup H_V \\
H_H &= B_1 \cup B_2 \cup ... \cup B_N \text{ donde } H = [0, 255], V = [50, 255], S = [50, 255] \\
H_S &= B_{N+1} \text{ donde } H = [0, 255], V = [0, 50], S = [50, 255] \\
H_V &= B_{N+2} \text{ donde } H = [0, 255], V = [50, 255], S = [0, 50]
\end{aligned}$$

Por tanto esto queda como:

$$H = \underbrace{B_1 \cup B_2 \cup ... \cup B_N}_{H=[0,255],V=[50,255],S=[50,255]} \cup \underbrace{B_{N+1}}_{H=[0,255],V=[0,50],S=[50,255]} \cup \underbrace{B_{N+2}}_{H=[0,255],V=[50,255],S=[0,50]} \tag{4.8}$$

Al final, el valor de cada clase se divide entre el número total de puntos para obtener los valores en porcentajes y hacer el histograma invariante a escala,

es decir, no importa si el objeto detectado tiene más o menos pixeles, si no la proporción de estos como parte de todo el objeto.

4.2.3. Error de color

El error de color se obtiene de la comparación entre los histogramas de entrenamiento guardados y el histograma de un objeto detectado, utilizando la *Intersección de Histogramas* propuesta por Swain and Ballard (1991) con el mismo propósito de reconocer objetos basados en color pero en el espacio de RGB.

La Intersección de Histogramas fue creada para trabajar utilizando equipos con procesamiento limitado sobre bases de entrenamiento grandes, por lo que se diseñó para que fuera muy eficiente, computacionalmente hablando. Además, este método se considera robusto para oclusión parcial, escala y rotaciónes del objeto.

Dados un par de histogramas I y M, cada uno con n número de clases, la Intersección de Histogramas Normalizada se define como:

$$e_{color} = H(I, M) = \frac{\sum_{j=1}^{n} \text{mín}(I_j, M_j)}{\sum_{j=1}^{n} M_j} \tag{4.9}$$

El rango de valores de esta métrica se encuentra entre $[0, 1]$, donde 1 representa el mejor valor, el cual corresponde al caso donde los histogramas son iguales. Asimismo, mientras este valor se reduzca, los histogramas tendrán una mayor diferencia hasta llegar a 0 donde, se dice que los histogramas son completamente diferentes. En este sentido, la Comparación de Histogramas no es un valor de error sino más bien una medida de similitud.

Experimentalmente, Swain and Ballard (1991) propusieron el valor de 0.6 como valor mínimo que debe arrojar la Comparación de Histogramas para decir que dos objetos son parecidos en color, para histogramas con $n = 64$ número de clases.

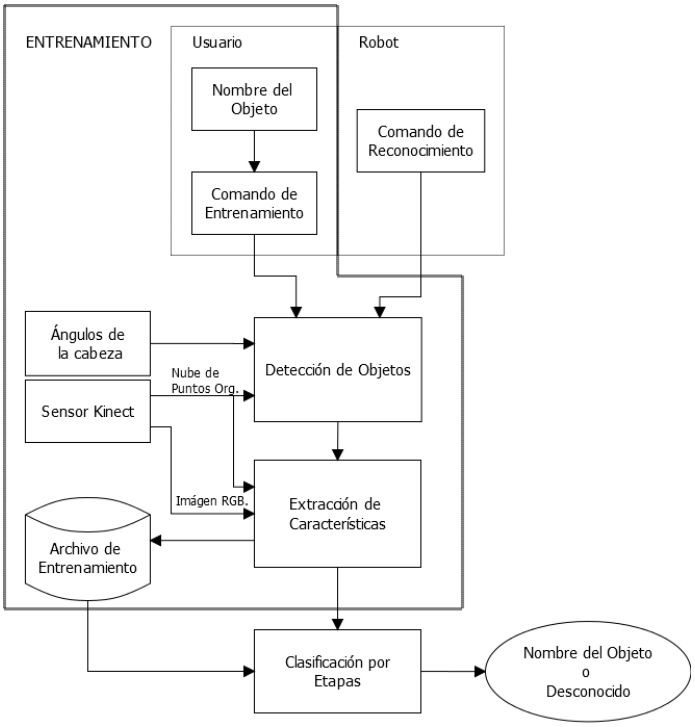


Figura 4.4: Arquitectura propuesta para el sistema.

4.3. Reconocimiento por etapas

Para entrenar y reconocer objetos, estos primero deben pasar por el proceso de detección, en el cual, se segmentan y se obtienen los índices para cada uno de ellos. La arquitectura general del sistema se puede observar en el diagrama de la figura 4.4.

4.3.1. Entrenamiento

El entrenamiento de los objetos se realiza utilizando diferentes capturas de este con una diferencia de rotación de 45 grados sobre el plano de la mesa entre cada una. A cada conjunto de capturas de un mismo objeto se le asigna un nombre, con el que será identificado el objeto y su conjunto de características

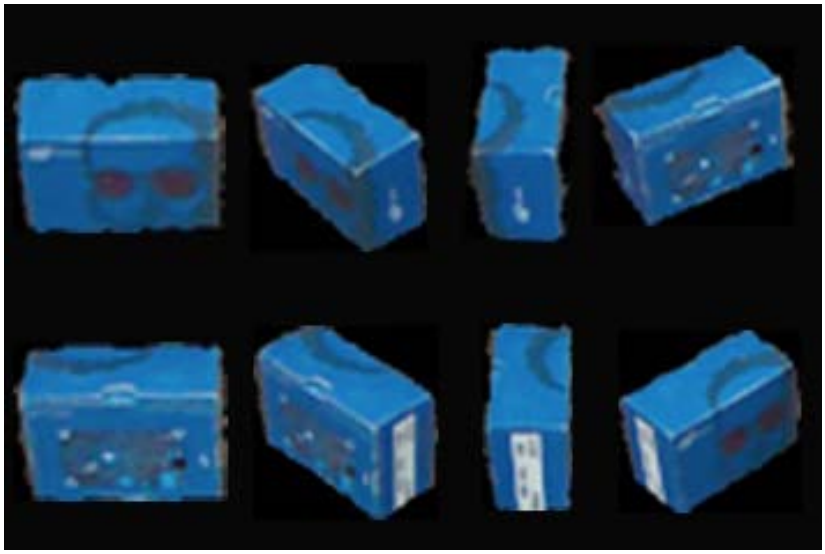


Figura 4.5: Ejemplo de capturas para el entrenamiento.

dentro del sistema. En la imagen 4.5 se puede observar un ejemplo de las capturas utilizadas para entrenar un objeto.

En cada captura, se extraen y se almacenan las cáracteristicas mencionadas en las secciones anteriores, las cuales son:

- Dimensión del objeto.
- Vector de Momentos Invariantes de Hu.
- Histograma de Color Normalizado.

Todos los conjuntos de características con sus respectivos nombres de objetos son almacenados en un archivo con formato .xml, el cual es leído cuando inicia el módulo de visión, por lo que en la fase de detección, ya residen en memoria, lo que acelera todo el proceso.

4.3.2. Reconocimiento

El método propuesto utiliza los errores propuestos, el de color, el de forma y el de tamaño, para decidir si un objeto detectado fue entrenado previamente y

clasificarlo como tal, o por el contrario, si este es un objeto desconocido, utilizando una discriminación de candidatos en tres etapas. En la figura 4.6 se puede ver un diagrama del clasificador propuesto.

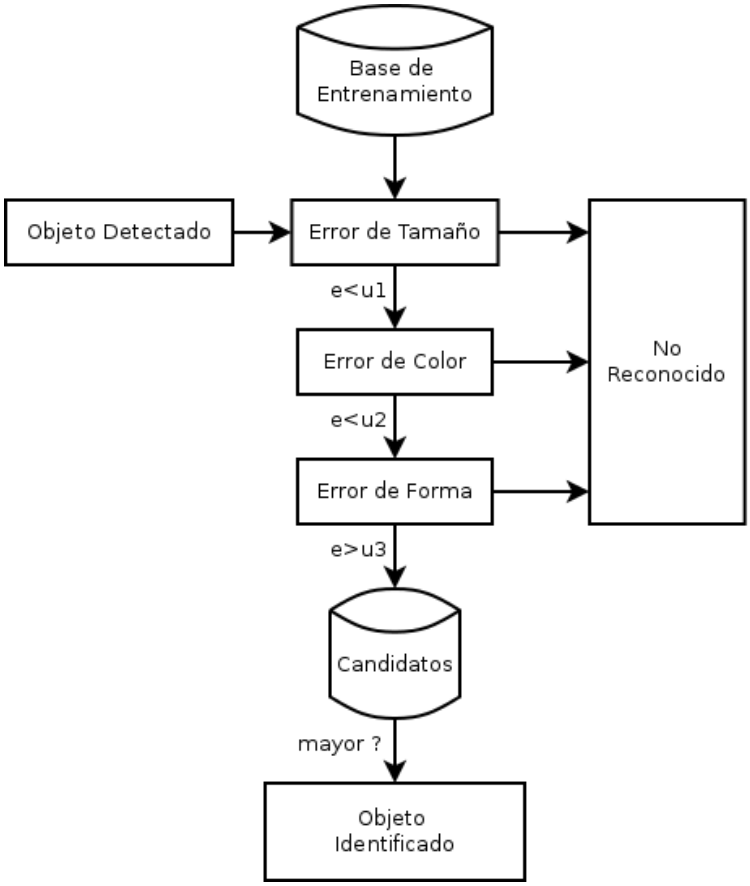


Figura 4.6: Diagrama del clasificador diseñado.

El clasificador diseñado extrae primero el tamaño de un objeto detectado y calcula el error de tamaño entre este y los tamaños de los objetos de la base de entrenamiento. Los objetos de entrenamiento cuyo error se encuentre por debajo de un umbral $u_{tamaño}$ son guardados para la siguiente etapa.

En la segunda etapa, el clasificador calcula el error de forma entre el objeto detectado y los objetos guardados durante la primera etapa. De la misma manera,

los objetos candidatos para los cuales, su error de forma sea menor a un cierto umbral u_{forma} son almacenados y pasan a la última etapa de clasificación.

La etapa final se basa en el error de color, donde los histogramas de los objetos que hayan sido almacenados durante las dos etapas anteriores son comparados con el histograma del objeto detectado utilizando la Intersección de Histogramas. Los errores de Histograma que se encuentren por arriba de un umbral u_{color} son ordenados de mayor a menor y se elige el mayor de estos como objeto reconocido.

Si en cualquiera de las etapas no existen objetos de entrenamiento para los que los errores pasen la prueba del umbral, se dice que el objeto detectado no ha sido reconocido. Además, en cada etapa se va reduciendo el número de candidatos por lo que el proceso se acelera.

Capítulo 5

Experimentos y resultados

5.1. Experimentos

Los experimentos realizados tratan de emular la prueba *Object Perception* de RoCKiN de 2014, en la que el robot debe identificar objetos de un conjunto determinado.

Durante la prueba se realizaron 20 tomas de cada objeto individualmente, ubicándolos en posiciones y orientaciones aleatorias en una mesa rectangular.

Se utilizaron 15 objetos para el entrenamiento. Estos fueron elegidos para simular los objetos presentados durante la competencia de RoCKiN 2014. Además, se seleccionaron grupos de objetos sin textura, con colores parecidos y formas y tamaños similares.

Los objetos utilizados y los índices utilizados para hacer referencia a estos en las tablas son:

(1) Bote Azul	(2) Bote Blanco	(3) Caja Azul
(4) Caja Café	(5) Caja Verde	(6) Portaretratos Café
(7) Portaretratos Dorado	(8) Lata Plateada	(9) Plato Verde
(10) Shampoo	(11) Taza Crema	(12) Taza Negra
(13) Taza Roja	(14) Tupper Azul	(15) Tupper Naranja

Tabla 5.1: Objetos utilizados para las pruebas.

Estos objetos se pueden ver en la figura 5.1.

Figura 5.1: Objetos utilizados durante los experimentos.

Los grupos propuestos son:

- Dos portaretratos de diferente tamaño y color.
  (Objetos 6 y 7)
- Dos recipientes de la misma forma pero diferente color.
  (Objetos 14 y 15)
- Tres cajas de la misma forma pero diferente color.
  (Objetos 3, 4, y 5)
- Tres botes de color y forma similar pero diferente tamaño.
  (Objetos 1,2 y 8)
- Tres tazas del tamaño y forma similar pero diferente color.
  (Objetos 11, 12 y 13)

- Dos objetos del mismo color pero diferente forma y tamaño.
  (Objetos 9 y 10)

Para todos los experimentos, las configuraciones del robot y las distancias respecto a la mesa fueron las mismas y tratan de ser parecidas a las del robot durante la competencia. En la figura 5.2 se observa una comparación entre los experimentos expuestos a continuación y el robot durante la competencia. Algunas de las caracteristísticas de la prueba son:

- Distancia del robot a la mesa: $0.50[m]$
- Ángulo de cabeceo (tilt) : $-40[deg]$
- Ángulo de guiñada (pan) : $0[deg]$
- Altura de la mesa: $1.2[m]$

5.1.1. Prueba 1: Reconocedor propuesto

Para esta prueba se utilizó el sistema descrito en los capítulos anteriores, el cual combina la forma, el tamaño y el color del objeto. En esta prueba se utilizó la altura del objeto como medida del tamaño del mismo.

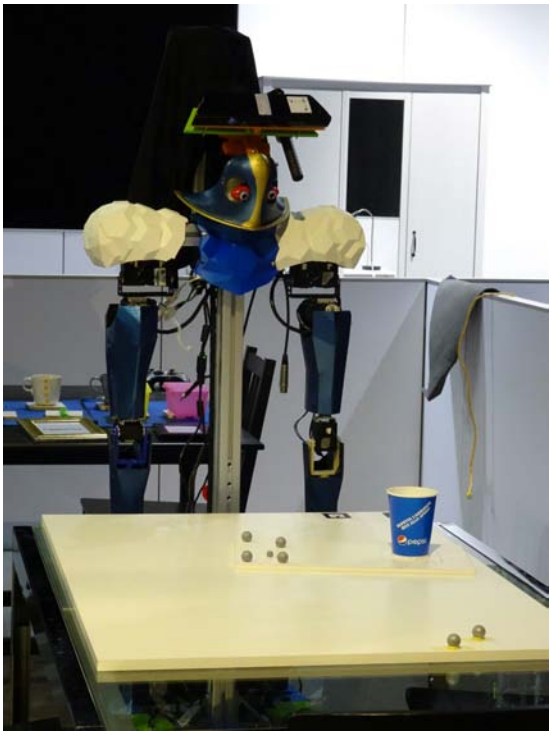
El tamaño de las clases para los histogramas se eligió en 18 para el Tono y 2 más para los blancos y los negros, es decir:

$$N_B = 20$$

Los umbrales, $u_{tamaño}$, u_{forma} y u_{color} obtenidos experimentalmente y utilizados durante esta prueba son:

$$u_{tamaño} = 0.01$$
$$u_{forma} = 0.1$$
$$u_{color} = 0.4$$

(a) Experimentos.



(b) RoCKIn 2014.

Figura 5.2: Robot durante la competencia y los experimentos.

Los resultados de esta prueba se pueden observar en la Matriz de confusión generada en la tabla 5.2, donde los renglones indican el objeto real utilizado, y las columnas los posibles resultados. El valor de la celda indica cuantas veces, el objeto fue reconocido como indica la columna. La columna *NR* significa "No Reconocido".

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	NR
1	14		-	-	-	-	-	-	-	-	-	-	-	-	-	6
2	-	20	-	-	-	-	-	-	-	-	-	-	-	-	-	0
3	-	-	20	-	-	-	-	-	-	-	-	-	-	-	-	0
4	-	-	-	20	-	-	-	-	-	-	-	-	-	-	-	0
5	-	-	-	-	19	-	-	-	-	-	-	-	-	-	-	1
6	-	-	-	-	-	17	-	-	-	-	-	-	-	-	-	3
7	-	-	-	-	-	-	10	-	-	-	-	-	-	-	-	10
8	-	-	-	-	-	-	-	20	-	-	-	-	-	-	-	0
9	-	-	-	-	-	-	-	-	18	-	-	-	-	-	-	2
10	-	-	-	-	-	-	-	-	-	19	-	-	-	-	-	1
11	-	-	-	-	-	-	-	-	-	-	20	-	-	-	-	0
12	-	-	-	-	-	-	-	-	-	-	-	20	-	-	-	0
13	-	-	-	-	-	-	-	-	-	-	3	-	17	-	-	0
14	-	-	-	-	-	-	-	-	-	-	-	-	-	20	-	0
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	20	0

Tabla 5.2: Matriz de confusión para el prueba 1.

En la tabla 5.3 se muestran los resultados por porcentajes de la prueba 1. En la columna de *Reconocidos* se muestra el porcentaje de aciertos del reconocedor. En la columna de *No Reconocidos* se establece el porcentaje de los objetos que no pasaron alguno de los 3 umbrales y el sistema no lo reconoció. La columna de *Errores*, muestra los resultados falsos, es decir, cuando el sistema confundió un objeto con otro.

Objeto	% Reconocido	% No reconocidos	% Errores
(1) Bote azul	70	30	0
(2) Bote blanco	100	0	0
(3) Caja azul	100	0	0
(4) Caja café	100	0	0
(5) Caja verde	95	5	0
(6) Portaretratos café	85	15	0
(7) Portaretratos dorado	50	50	0
(8) Lata plateada	100	0	0
(9) Plato verde	90	10	0
(10) Shampoo	95	5	0
(11) Taza crema	100	0	0
(12) Taza negra	100	0	0
(13) Taza roja	85	0	15
(14) Tupper azul	100	0	0
(15) Tupper naranja	100	0	0
TOTAL	**91.333**	**7.666**	**1.000**

Tabla 5.3: Resultados por porcentajes de la prueba 1.

El tiempo promedio de detección y reconocimiento de cada captura para la prueba 1 fue de:

$$\tau_{avg} = 0.283[s]$$

5.1.2. Prueba 2: Sistema actual basado en SIFT.

Para comparar los resultados obtenidos se realizó el mismo proceso, 15 objetos y 20 capturas para cada uno, utilizando el sistema actual implementado en el módulo de Vision del robot, el cual se basa en la extracción y descripción de puntos de interés *SIFT* y una verificación de estos utilizando el algoritmo de RANSAC para encontrar una homografía que valide la transformación de los puntos de interés del entrenamiento y los puntos de interés del objeto detectado, como indica Lowe (2004).

Para el entrenamiento, se utilizaron las mismas 8 imágenes (ver figura 4.5) de cada objeto que se utilizaron durante la extracción del Histograma del Color para el entrenamiento del sistema propuesto.

Los resultados de esta prueba se sintetizaron en la tabla 5.4, donde se generó la Matriz de confusión que muestra el objeto identificado para cada captura.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	NR
1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	20
2	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	20
3	-	-	18	-	-	-	-	-	-	-	-	-	-	-	-	2
4	-	-	-	20	-	-	-	-	-	-	-	-	-	-	-	0
5	-	-	-	-	16	-	-	-	-	-	-	-	-	-	-	4
6	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	20
7	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	19
8	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-	15
9	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	20
10	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	20
11	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	20
12	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	18
13	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	16
14	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	20
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	20

Tabla 5.4: Matriz de confusión para el prueba 2.

De la misma manera, en la tabla 5.5 se muestran los porcentajes de reconocimiento, de no reconocimiento y de errores.

Objeto	% Reconocido	% No reconocidos	% Errores
(1) Bote azul	0	100	0
(2) Bote blanco	0	100	0
(3) Caja azul	90	10	0
(4) Caja café	100	0	0
(5) Caja verde	85	15	0
(6) Portaretratos café	0	100	0
(7) Portaretratos dorado	5	95	0
(8) Lata plateada	25	75	0
(9) Plato verde	0	100	0
(10) Shampoo	0	100	0
(11) Taza crema	0	100	0
(12) Taza negra	10	90	0
(13) Taza roja	20	80	0
(14) Tupper azul	0	100	0
(15) Tupper naranja	0	100	0
TOTAL	**22.33333333**	**77.66666667**	**0**

Tabla 5.5: Resultados por porcentajes de la prueba 2.

El tiempo promedio de detección y reconocimiento de cada captura para la prueba 2 fue de:

$$\tau_{avg} = 0.903[s]$$

5.2. Análisis de los resultados.

Como se puede observar en las tablas 5.3 y 5.5, el algoritmo propuesto presentó mejoras significativas en cuanto a la detección y tiempos de ejecución. En la tabla 5.6 se muestran los resultados totales, remarcando los mejores valores.

Sistema	% Reconocido	% No reconocidos	% Errores	τ_{avg}
Propuesto	**91.333**	**7.666**	1.000	**0.283**
Actual	22.33333333	77.66666667	**0.000**	0.903

Tabla 5.6: Comparación de los resultados totales.

La disminución de los tiempos de ejecución se deben en parte a que los vectores de características propuestos son de una dimensionalidad menor que el utilizado por SIFT. Mientras que el sistema propuesto se basa en 28 características [1] por imagen, el sistema SIFT puede encontrar hasta cientos de puntos caracteísticos con descriptores de dimensión 128, lo que aumenta el tiempo de búsqeda y comparación.

Estos resultados eran esperados para los objetos sin textura, sin embargo, también hubo mejorías en diversos objetos con textura, como en el caso del objeto (8). Esto se debe en parte a que, dado que el objeto es cilíndrico y la homografía se basa en la proyección de puntos entre planos, la transformación no validó los puntos. Para evitar esta situación, cuando se utilizó el sistema actual, los objetos eran entrenados más veces, con giros menores entre cada toma, para compensar la forma de los objetos.

Usando el sistema propuesto, los porcentajes de la mayoría de los objetos supera el 85 % de reconocimiento. Más del 50 % de los objetos, es decir 8 de 15 objetos, se reconocieron correctamente el 100 % de las veces.

A pesar de los buenos resultados, analizando los datos se puede ver que existe una ventaja importante utilizando la versión actual basada en SIFT frente a la propuesta en el presente trabajo. El reconocedor basado en SIFT no presentó errores, es decir, no confundió objetos. A partir de las imágenes obtenidas durante el desarrollo de la prueba 1, se analizaron los 3 errores que presentó la taza roja (13) y se determinó que existe una configuración en la que el mango mira directamente a la cámara y el porcentaje de rojos que se visualiza es mínimo. Por esto y dada la baja resolución de las imágenes, el sistema presentó errores y confundió al objeto con otro muy parecido, si se visualiza desde este mismo

[1] 1 para la altura, 7 para los Momentos de Hu y 20 para el Histograma de Color.

ángulo. A esto se le denominó error por falta de información de color. En la figura 5.3 se pueden observar las disposiciónes de la taza que originaron las 3 tomas erróneas.

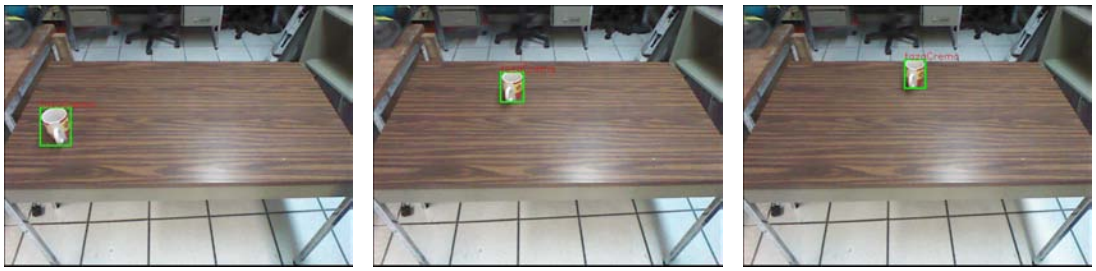


Figura 5.3: Errores debidos a la falta de información de color.

De los errores presentados para el sistema propuesto, tres se atribuyen a una mala segmentación (objetos 5 y 9). Uno debido a la falta de información por parte del Kinect en los bordes de la nube de puntos organizada y dos debidos al ruido presentado cuando los objetos están muy cerca del kinect y al centro, presentando una falta de información de profundidad debido a que el sensor IR y el emisor IR del kinect se encuentran separados una cierta distancia. En la figura 5.4 se observan las capturas de los errores originados por falta de información 3D.

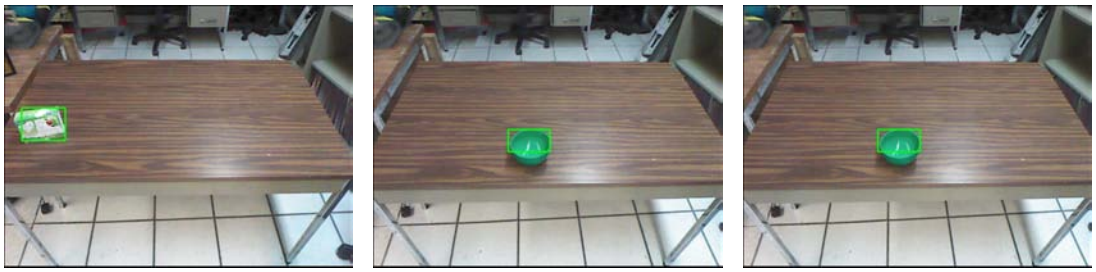


Figura 5.4: Errores debidos a una mala segmentación por falta de información 3D.

El objeto que más errores presentó durante la identificación fue el Portaretratos dorado (7), el cual solo se pudo reconocer un 50 % de las veces. De igual manera, analizando las capturas de las pruebas, se observó que las veces que

no se pudo identificar fue cuando este se encontraba visto de frente. Al analizar los datos, también se observó que el umbral que detenía el reconocimiento fue el umbral de forma. El polígono generado por la proyeccion de sus puntos en el plano, dado que su forma visto desde enfrente es un rectángulo casi vertical, es un rectángulo con dos de sus lados muy angostos. Por tanto, se dedujo que figuras muy angostas, no generan Momentos de Hu muy confiables.

Los demás errores se atribuyen simplemente al ruido del sensor y a los cambios de iluminación durante el entrenamiento y las pruebas. Sin embargo, aún cuando el reconocedor no fue capaz de identificar el 100 % de las veces, de 300 capturas de diferentes objetos reconoció 274.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

Utilizando la información que proporciona el sensor Kinect, es decir, la imagen RGB y la nube de puntos organizada, se logró diseñar un algoritmo que permite detectar los objetos que se encuentran sobre planos horizontales.

Este algoritmo hace uso de la técnica para estimar modelos RANSAC. Aunque RANSAC es un método iterativo, el cual puede ser utilizado para extraer planos de nubes de puntos original, en la implementación propuesta, se calcula nueva información que ayuda a que la convergencia del algoritmo sea más eficiente. La información que permite esta rápida convergencia son las normales con dirección vertical, las cuales son rápidamente identificadas mediante la extracción de normales locales y la transformación de la nube de puntos, utilizando la información extra que nos proporcionan los ángulos de la cabeza del robot.

Además de esto, se consiguió segmentar los objetos que se encuentran a cierta distancia uno del otro. Con esto, es posible, utilizando la misma toma, reconocer diferentes objetos al mismo tiempo, reduciendo el tiempo total de ejecución del algoritmo. Todo esto se realizó utilizando únicamente la nube de puntos organizada, es decir la información directa del Kinect, y no se necesitaron pasos

intermedios para crear estructuras para el manejo de nubes de puntos, como pueden ser árboles k-dimensionales.

Una buena segmentación de los objetos permite utilizar técnicas muy diversas para su caracterización. Sin este, las técnicas propuestas para lograr el reconocimiento de objetos no se hubieran podido utilizar. Para esto, la nube de puntos organizada continene mucha información que puede ser explotada para realizar una adecuada detección y segmentación.

En cuanto a la identificación de objetos con poca o nula textura, con el sistema propuesto se consiguió mejorar el porcentaje de reconocimiento, logrando subir este porcentaje de un 22.333 % a un 91.333 %, es decir, aproximadamte 4 veces. Esta mejora no representó un aumento en los tiempos de ejecución, sino que al contrario, los tiempos de ejecución se redujeron a un 32 % de los que conseguían con el sistema anterior.

El tamaño de un objeto es una característica muy distintiva del mismo. De hecho, para las pruebas, fue dificil encontrar objetos de uso común que tuvieran las mismas dimensiones en las tres componentes que lo definen, es decir, largo, ancho y alto. El Kinect, a pesar del ruido que contine, es un buen sensor para estimar el tamaño de los objetos, por lo que este sensor se puede utilizar como una manera muy eficiente de reducir candidatos para identificar objetos de manera rápida.

Se propuso una nueva técnica para hacer el reconocimiento de objetos 3D a partir de la proyección de los puntos en 2D utilizando los Momentos Invariantes de Hu. Esta nueva técnica, combinada con otras carácteristicas que todo objeto en nuestro alrededor contienen, como lo son el tamaño y el color, pueden servir para describir con precisión objetos comúnes cuando otras técnicas muy utilizadas fallan, como lo es el método de SIFT.

El color es otro aspecto muy distintivo de los objetos y existen otros espacios de color en donde esta propiedad aporta mejores características para usar en el reconocimiento. Además, se comprobó experimentalmente que la técnica denominada Intersección de Histogramas es muy adecuada para comparar dos Histogramas, por lo que es de gran utilidad para el reconocimiento de objetos.

En general, se puede decir que la hipótesis se validó completamente, puesto que, mediante pruebas experimentales, se determinó que, utilizando el sistema propuesto, el reconocimiento de objetos con poca o nula textura mejoró considerablemente si lo comparamos con el sistema actual, basado en la extracción de puntos de interés.

6.2. Trabajo a futuro

En esta sección se pretende dar ideas que permitan continuar con el desarrollo de este trabajo. Ideas que, por tiempo o por encontrarse fuera de la línea que seguía el presente trabajo, no pudieron ser probadas y estudiadas exhaustivamente.

Durante el presente trabajo, se trató de enfatizar el considerable ruido que presenta el sensor Kinect cuando se trabaja con la información 3D del mismo. Actualmente, existe en el mercado la segunda versión de este sensor, en la cual, para obtener información de los objetos se utiliza una sensor del tipo "Tiempo de Vuelo". Este último proporciona considerablemente menos ruido que su predecesor y la resolución, tanto de la nube de puntos como de la imagen RGB, es mayor. Dada la complejidad de los algoritmos presentados, es de suponer que estos se comportarían mejor utilizando este sensor y los tiempos de ejecución, dado el incremento de información que conlleva el aumento en la resolución, se mantendrían equiparables.

En cuanto a la detección de objetos, utilizando el algoritmo de segmentación propuesto, estos deben estar a una cierta distancia uno del otro, de lo contrario se corre el riesgo de obtener una mala segmentación, cuando los objetos son concavos (como el plato de las pruebas). Utilizar otros algoritmos de segmentación podría suponer una mejoría en cuanto a las restricciones del algoritmo. Asimismo, los objetos deben de estar sobre planos horizontales para su detección, implementar algoritmos que permitan encontrar objetos sobre otras superficies, inclusive si estos no se encuentran sobre una superficie, como puede ser en las manos de una

persona, supone un reto difícil pero de lograrse, sortearía una de las principlaes desventajas de este método.

La parte de reconocimiento de objetos también plantea algunos caminos para su mejora. En la implementación propuesta, se utlizaron los Momentos Invariantes de Hu para identificar los polígonos de la proyección de los puntos en un plano, sin embargo, existen otras técnicas para comparar formas en 2D, las cuales faltan por ser probadas. Igualmente, esta proyección podría hacerse en otros planos, como pueden ser todas las caras de la caja delimitadora. Esto permitiría al algoritmo ser invariante a la rotación sobre más ejes de referencia. De la misma manera, un estudio sobre diferentes espacios de color y formas de comparar los histogramas parece un camino obvio para continuar con el reconcimiento basado en color.

Apéndice A

Código fuente

A.1. Cálculo del tamaño y Momentos de Hu

```
DetectedObject( std::vector< cv::Point2i > indexes, cv::Mat xyzPoints, cv::Mat
    oriMask, PlanarHorizontalSegment planarSeg ){
        this->centroid = cv::Point3f(0.0, 0.0, 0.0);
        this->indexes = indexes;
        this->xyzPoints = xyzPoints;
        this->oriMask = oriMask;
        this->planarSeg = planarSeg;
        this->height = 0;
        for( int i=0; i<indexes.size(); i++){
                cv::Point3f pXYZ = xyzPoints.at<cv::Vec3f>( indexes[i] );
                this->pointCloud.push_back( pXYZ );
                this->xyPoints2D.push_back( cv::Point2f(pXYZ.x, pXYZ.y) );
                double distToPlane = this->planarSeg.DistanceToPlane( pXYZ );
                if( distToPlane > this->height )
                        this->height = distToPlane;
                this->centroid = this->centroid + pXYZ;
        }
        this->centroid = this->centroid * ( 1.0 / (float)indexes.size() );
        this->boundBox = cv::boundingRect( this->indexes );
        this->shadowOriBoundBoxt2D = cv::minAreaRect( xyPoints2D );
        cv::convexHull( xyPoints2D, this->shadowCHull );
        cv::approxPolyDP( this->shadowCHull, this->shadowContour2D, 0.00001,
            true);
        this->moments = cv::moments( this->shadowContour2D , bool);
        cv::HuMoments( this->moments, this->hu);
}
```

A.2. Extracción de histogramas

```
cv::Mat ColorObjectRecognizer::CalculateHistogram( cv::Mat bgrImage, cv::Mat
    mask )
{
        cv::Mat hsvImage;
        cv::cvtColor( bgrImage, hsvImage, CV_BGR2HSV_FULL );
        cv::Mat histogram;
        int chan[] = { 0 };
        int histSize[] = { binNo };
        float hueRange[] = { 0, 255 };
        const float* ranges[] = { hueRange };

        // Hue
        cv::Mat satValMask;
        cv::inRange( hsvImage, cv::Scalar( 0, 50, 50), cv::Scalar(255, 255, 205)
            , satValMask);
        cv::Mat maskAnd = mask & satValMask;

        // Blacks
        cv::Mat satM;
        cv::inRange( hsvImage, cv::Scalar( 0, 0, 0), cv::Scalar(255, 50, 50),
            satM);
        cv::Mat maskAnd_1 = mask & satM;
        int blackPx = cv::countNonZero( maskAnd_1 );

        // Whites
        cv::Mat valM;
        cv::inRange( hsvImage, cv::Scalar( 0, 0, 205), cv::Scalar(255, 50, 255),
             valM);
        cv::Mat maskAnd_2 = mask & valM;
        int whitePx = cv::countNonZero( maskAnd_2 );
        calcHist(&hsvImage, 1, chan, maskAnd, histogram, 1, histSize, ranges,
            true, false);

        cv::Mat newHisto = cv::Mat( histogram.rows+2, histogram.cols, histogram.
            type() );
        for( int i=0; i< histogram.rows; i++)
                newHisto.at<float>(i,0) = histogram.at<float>( i, 0);
        newHisto.at<float>( histogram.rows , 0 ) = blackPx;
        newHisto.at<float>( histogram.rows+1 , 0 ) = whitePx;

        cv::normalize(newHisto, newHisto, 1.0, 0.0, cv::NORM_L1);
        return newHisto;
}
```

A.3. Clasificación por etapas

```
std::pair<double, std::string> RecognizeObject( std::vector< std::pair< double,
    std::string> > sizeError , std::vector< std::pair< double, std::string> >
    shapeError , std::vector< std::pair< double, std::string> > colorError ,
    double& outErrorSize , double& outErrorShape , double& outErrorColor)
{
        double sizeThres = 0.01;
        double shapeThres = 0.1;
        double colorThres = 0.4;
        std::pair<double, std::string> bestSoFar(-1000.00, "");
        for( int i=0; i<sizeError.size(); i++)
        {
                bool sizePass = sizeThres > sizeError[i].first;
                bool shapePass = shapeThres > shapeError[i].first;
                bool colorPass = colorThres < colorError[i].first;
                if( sizePass && shapePass && colorPass )
                {
                        if(colorError[i].first > bestSoFar.first)
                        {
                                bestSoFar.first = colorError[i].first;
                                bestSoFar.second = colorError[i].second;

                                outErrorSize = sizeError[i].first;
                                outErrorShape = shapeError[i].first;
                                outErrorColor = colorError[i].first;
                        }
                }
        }
        return bestSoFar;
}
```

Bibliografía

Bradski, G. and Kaehler, A. (2008). *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media.

Endres, F., Hess, J., Engelhard, N., Sturm, J., Cremers, D., and Burgard, W. (2012). An evaluation of the rgb-d slam system. In *Robotics and Automation (ICRA), 2012 IEEE International Conference*, pages 1691–1696. IEEE.

Falahati, S. (2013). *OpenNI Cookbook*. Packt Publishing.

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

Flusser, J. and Suk, T. (2006). Rotation moment invariants for recognition of symmetric objects. *IEEE Transactions on Image Processing*, 15(12):3784–3790.

Freeman, H. and Shapira, R. (1975). Determining the minimum-area encasing rectangle for an arbitrary closed curve. *Commun. ACM*, 18(7):409–413.

Gevers, T. and Smeulders, A. W. (1999). Color-based object recognition. *Pattern Recognition*, 32(3):453 – 464.

Graham, R. L. and Yao, F. F. (1983). Finding the convex hull of a simple polygon. *Journal of Algorithms*, 4(4):324–331.

Hast, A., Nysjö, J., and Marchetti, A. (2013). Optimal ransac-towards a repeatable algorithm for finding the optimal set.

Holz, D., Holzer, S., Rusu, R. B., and Behnke, S. (2012). Real-time plane segmentation using rgb-d cameras. In *RoboCup 2011: Robot Soccer World Cup XV*, pages 306–317. Springer.

Hu, M.-K. (1962). Visual pattern recognition by moment invariants. *Information Theory, IRE Transactions on*, 8(2):179–187.

Khoshelham, K. (2011). Accuracy analysis of kinect depth data. In *ISPRS workshop laser scanning*, volume 38, page W12.

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

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110.

Mehtre, B. M., Kankanhalli, M. S., and Lee, W. F. (1997). Shape measures for content based image retrieval: a comparison. *Information Processing & Management*, 33(3):319–337.

Microsoft (2010). Kinect for windows sensor components and specifications. `https://msdn.microsoft.com/en-us/library/jj131033.aspx`. Accesado el 17-04-2015.

O'Rourke, J. (1985). Finding minimal enclosing boxes. *International journal of computer & information sciences*, 14(3):183–199.

Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 6*, 2(11):559–572.

RoCKIn (2015). Rockin @ home. `http://rockinrobotchallenge.eu/home.php`. Accesado el 22-11-2015.

Savage-Carmona, J., Billinghurst, M., and Holden, A. (1998). The virbot: a virtual reality robot driven with multimodal commands. *Expert Systems with Applications*, 15(3):413–419.

Swain, M. J. and Ballard, D. H. (1991). Color indexing. *International journal of computer vision*, 7(1):11–32.

Weingarten, J. W., Gruener, G., and Siegwart, R. (2004). Probabilistic plane fitting in 3d and an application to robotic mapping. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pages 927–932. IEEE.