ESCUELA POLITÉCNICA

UNIVERSIDAD DE EXTREMADURA

ESCUELA POLITÉCNICA

GRADO EN INGENIERÍA INFORMÁTICA EN INGENIERÍA DE COMPUTADORES

TRABAJO FIN DE GRADO

Reconocimiento de objetos y obtención de su posición.

ESCUELA POLITÉCNICA

UNIVERSIDAD DE EXTREMADURA

ESCUELA POLITÉCNICA

GRADO EN INGENIERÍA INFORMÁTICA EN INGENIERÍA DE COMPUTADORES

TRABAJO FIN DE GRADO

Reconocimiento de objetos y obtención de su posición.

Autor: Iván Barbecho Delgado

Tutor: Pablo Bustos Garcia Castro

Co-Tutor: Luis J. Manso Fernández-Argüelles

Resumen

Este proyecto se ha desarrollado con el objetivo principal de dotar a Shelly, el robot asistencial de RoboLab, de un sistema de reconocimiento y localización de objetos en el espacio. Este documento consta de tres partes: una introductoria que comprende los capítulos 1 y 2, una teórica que se desarrolla en el capítulo 3 y una práctica que se expone en los capítulos 4, 5 y 6.

En la sección teórica se recogen los conceptos necesarios para la comprensión de los diferentes algoritmos utilizados en la implementación; distintos tipos de descriptores de nubes de puntos; diferentes técnicas de ajuste de nubes de puntos; extracción de los objetos de una escena; cómo es Shelly; qué es RoboComp y la arquitectura cognitiva CORTEX.

En la sección práctica se desarrolla el proceso seguido para implementar el sistema objetivo: cómo se debe realizar el entrenamiento del sistema; qué información ha de almacenarse; cómo corregir la pose calculada en el entrenamiento; cómo se realiza el reconocimiento de los objetos; cómo se localiza un objeto en el espacio y qué información debe devolver el sistema; qué se puede hacer para optimizar el sistema; y finalmente, cómo combinar el reconocimiento y la localización del objeto. Además de esto, se muestran los resultados obtenidos de los experimentos de reconocimiento de objetos (probando los tipos de descriptores y los diferentes tipos de entrenamiento del sistema) y de localización del objeto (probando las diferentes técnicas de ajuste de nubes de puntos y los distintos tipos de entrenamiento del sistema).

Finalmente, se presenta un análisis detallado de los resultados de los experimentos llevados a cabo, llegándose a la conclusión de que la elección del tipo descriptor y de la técnica de ajuste influyen significativamente en los resultados que obtiene el sistema, así como en el tiempo de ejecución. La combinación del descriptor OUR-CVFH, 16 vistas de cada objeto y ajuste utilizando ICP, a la vista de los resultados obtenidos, ofrece mejores resultados dado que OUR-CVFH junto con 16 vistas de referencia tienen una menor tasa de errores que las demás configuraciones. ICP además de tener

un menor tiempo de ejecución, tiene una mayor probabilidad de devolver la pose correcta. La memoria del trabajo concluye con algunas ideas sobre posibles mejoras a desarrollar en el futuro.

Abstract

This project has been developed with the main objective of providing Shelly, the assistant robot of RoboLab, with an object recognition and localization system. This document is composed of three parts: an introductory part that covers chapters 1 and 2, a theoretical part which is explained in chapter 3, and a practical part comprising chapters 4, 5 and 6.

In the theoretical section the necessary concepts for understanding the algorithms used are introduced; various types of point cloud descriptors; the different points cloud fitting techniques; how are the objects from a scene segmented; a description of the robot Shelly; RoboComp and the CORTEX cognitive architecture.

The practical section presents the process followed for implementing the target system: how should system training be done; what information is to be stored; how to correct the computed pose in training; how objects are recognized; how an object is located in space and which information the system must return; what can be done to optimize the system; and finally how to combine the recognition and location of the object. In addition to this, the chapter presents the results obtained from the object recognition experiments (benchmarking the different types of descriptors and the different types of training system) and the experiments for location of the object (benchmarking the different techniques for fitting point clouds and different types of training systems).

Finally, a detailed analysis of the results of the experiments is presented, concluding that the choice of the descriptor type and the adjustment technique influence significantly the results obtained by the system, as well as at the execution time. The combination of the OUR-CVFH descriptor, 16 views of each object and fitting using ICP, in view of the results obtained, it offers better results given that OUR-CVFH together with the 16 points of view, obtain a lower error rate than the other configurations and ICP, in addition to having a shorter execution time, has a greater probability of returning the correct pose. The memory of the work concludes with some ideas on possible improvements to be developed in the future

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Los seres humanos esperan, a medio plazo, poder confiar muchas de las tareas que actualmente realizan de forma cotidiana a robots. Para que esto sea realidad en un futuro, es necesario que los robots puedan recabar información sobre los objetos de su entorno con suficiente precisión como para manipularlos.

Los seres humanos, a diferencia de los robots, tienen una capacidad extraordinaria para reconocer objetos, incluso cuando nunca han visto una determinada combinación de color, tamaño y textura. Para entrar en situación, imaginemos un robot al que se le ha asignado la tarea de coger una taza de una mesa. La principal pregunta que se plantea es ***¿Cómo puede el robot reconocer la taza?*** La respuesta a esta pregunta suele implicar la utilización de un sistema de reconocimiento de objetos, pero existen diversos tipos con características muy diferentes. En primer lugar, los sistemas más habituales de detección de objetos son los que dependen de las técnicas de coincidencia de características 2D, pero no funcionan suficientemente bien cuando están sujetos a diferentes condiciones de iluminación, textura u orientación de la cámara. Estos problemas pueden ser solventados con una segunda opción, los sistemas de detección de objetos 3D en los cuales se utilizan sensores RGBD, que estiman la distancia a la escena de cada píxel además de ofrecer una imagen en color. Además de la detección, para que el robot pueda coger finalmente la taza objetivo, se necesita conocer la posición en la que ésta se encuentra con una precisión suficiente. Aquí surge otra

pregunta, ***¿cómo puede el robot localizar la taza en el espacio?*** En este caso, es mucho más difícil localizar un objeto con un sistema 2D que no capte la profundidad de la escena. Con sistemas 3D, sin embargo, esta pregunta se puede resolver eficientemente con diferentes técnicas. En este proyecto se localizan los objetos en el espacio mediante técnicas basadas en el ajuste de nubes de puntos.

En este documento se explica la teoría necesaria y los pasos a seguir para implementar un sistema de reconocimiento de objetos y obtención de su posición, obteniendo unos resultados fiables. Finalmente, se presentan experimentos del sistema implementado realizados sobre el robot Shelly, el robot asistencial de RoboLab [1], para poder comparar las distintas configuraciones y obtener criterios para seleccionar la más adecuada.

En el siguiente capítulo se exponen los objetivos que se han marcado para realizar la implementación del sistema de reconocimiento de objetos y obtención de su posición.

Capítulo 2

Objetivos

El objetivo principal de este proyecto es dotar a Shelly, el robot asistencial de RoboLab, de un sistema de reconocimiento visual con el que pueda localizar un objeto en el espacio y obtener su pose[1] para posteriormente cogerlo. Para lograr este objetivo se han marcado las siguientes metas específicas:

- Capturar la nube de puntos de la escena que ve el robot usando una cámara RGBD, en concreto una Asus Xtion PRO Live.
- Detectar y suprimir de la nube de puntos capturada aquellos puntos pertenecientes al plano formado por el tablero de la mesa, así como aquellos que están por debajo del plano de la mesa y aquellos que no se sitúan sobre ella.
- Segmentar la nube de puntos resultante, separándola en diferentes nubes de puntos que correspondan a los distintos objetos que haya sobre la mesa.
- Obtener para cada objeto resultante, su nube de puntos y su pose.
- Estudiar los diferentes descriptores de objetos 3D usados en sistemas de detección basados en nubes de puntos: VFH, CVFH, OURCVFH.
- Reconocer cada uno de los objetos situados sobre una mesa que hayan sido previamente segmentados.

[1] Se utilizará el término pose para denotar la posición y orientación del objeto en el espacio.

- Estudiar las diferentes técnicas de ajuste de nube de puntos, necesarias para estimar la pose de los objetos detectados.
- Obtener la pose de los objetos de la escena que se hayan reconocido y etiquetado.
- Obtener el mínimo cubo delimitador alienado con los ejes (*axis-aligned minimum bounding box*) que contenga a la nube de cada objeto de la escena.
- Optimizar el sistema para obtener unos resultados fiables, reduciendo en la medida de lo posible el tiempo de cómputo.

Capítulo 3

Antecedentes

Con el paso de los años las técnicas de detección y reconocimiento de objetos han ido evolucionando debido, en gran parte, a la necesidad de la industria de automatizar los procesos de identificación, clasificación y manipulación. Estos sistemas se aplican rutinariamente en las cadenas industriales de producción para procesar productos. También se han comenzado a utilizar en robots sociales para interaccionar con personas.

El objetivo de conseguir un sistema de detección y reconocimiento de objetos puede ser logrado mediante diferentes tipos de sistemas. El primero utilizando sistemas 2D, que a su vez puede utilizarse dos técnicas distintas, **sistema de reconocimiento mediante la comparación de las características**, obteniéndose las características mediante unos algoritmos (*SIFT, SURF, ORB...*) para un conjunto de imágenes, posteriormente para una escena determinada se calcula las características y se compara con las características de la batería de imágenes; **aprendizaje profundo** (*deep learning*) está compuesto por un conjunto de algoritmos de aprendizaje automático que intenta modelar abstracciones de nivel alto en datos, utilizando arquitecturas compuestas de transformaciones no lineales múltiples.

Estas técnicas son poco fiables ya que en escenas de poca luminosidad obtendremos resultados erróneos. Además, si sólo disponemos de una imagen de la escena, no podremos localizar dicho objeto en el espacio. De esta manera surgen los sistemas de

reconocimiento de objetos 3D, en los cuales se utilizan alternativas como iluminar la escena con infrarrojos para obtener los puntos 3D de la escena, solventado el problema de la luminosidad y el posicionamiento de objetos, gracias a que contamos con la distancia a los puntos de la escena.

Tanto los sistemas 2D de obtención de características más sistema de comparación como los 3D utilizan descriptores de los objetos, almacenando en estos las características con las que posteriormente se realizará la comparación y cálculo del más semejante. Estos descriptores se pueden obtener atendiendo a diferentes aspectos: la apariencia, los bordes, la curvatura, etc. En los sistemas 2D de aprendizaje profundo los descriptores se calculan en la misma red.

En la actualidad existen una gran variedad de descriptores de nubes de puntos con los cuales se pueden aplicar las técnicas de *matching*. A continuación, se explicarán diferentes tipos, pero para entender estos tipos necesitará conocer previamente algunos conceptos clave:

3.1. Conceptos clave

3.1.1. Nube de Puntos

Es un conjunto digital de puntos en un sistema de coordenadas tridimensional que representan a un conjunto homólogo en la escena. Estos puntos se identifican habitualmente como coordenadas X, Y, y Z. Las nubes de puntos se suelen capturar con cámaras RGBD o con láser-escáner. A continuación, se muestra un ejemplo (3.1) y (3.2) correspondiendo éstas a la misma captura, desde diferentes ángulos (frente y rotada).

Figura 3.1: Nube de puntos sombrero de frente

Figura 3.2: Nube de puntos sombrero rotada

3.1.2. Entorno de vecindad

Siendo **P** una nube de puntos, se puede definir el entorno de vecindad de un punto $p_q \in P$ con radio r como el conjunto de puntos $p_i \in P$ donde $||p_q - p_i|| \leq d_r, \forall i \neq q$. En la figura 3.3 los puntos rojos representan al conjunto de puntos del entorno de vecindad de p_q, mientras que los puntos azules representan los puntos que son excluidos de este.

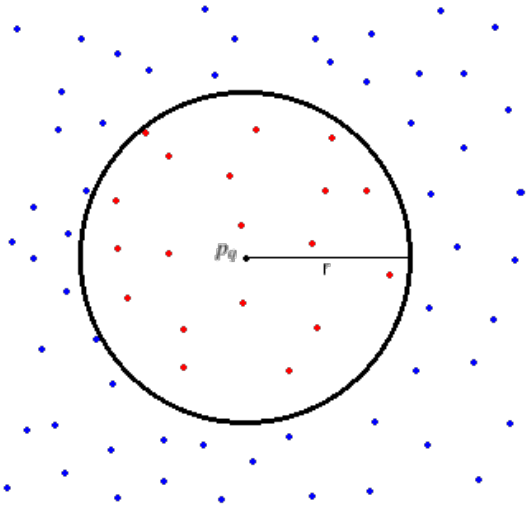

Figura 3.3: Representación del entorno de vecindad de un punto p_q con radio r

3.1.3. Entorno de vecindad de doble radio

Para un punto $p_q \in P$ se definen dos radios r_1 y r_2 siendo $r_1 < r_2$, definiéndose así dos entornos de vecindad concéntricos $P^{r1} y P^{r2}$, obteniendo dos formas distintas de representar p_q. P^{r1} esta formado por el conjunto de puntos $p_i \in P \; ||p_q - p_i|| \leq d_{r1}, \forall i \neq q$ y P^{r2} esta formado por el conjunto de puntos $p_j \in P \; ||p_q - p_j|| \leq d_{r2}, \forall j \neq q$, de esta manera se puede decir que P^{r2} contiene los puntos de P^{r1}. Ver figura 3.4

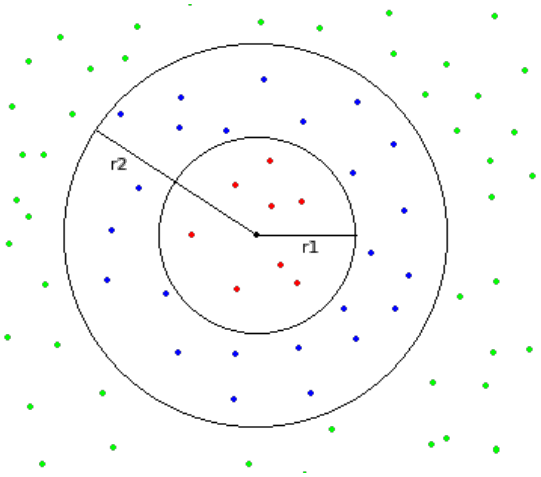

Figura 3.4: Representación del entorno de vecindad de doble radio de un punto, P^{r1} lo forman los puntos rojos, P^{r2} lo forman los puntos rojos y azules

3.1.4. Métricas entre puntos de superficie

En las nubes de puntos es imprescindible definir una métrica para medir la distancia entre los puntos de ésta. A continuación, se definen las métricas más comunes, tomando la siguiente premisa.

Siendo $\vec{u}$ y $\vec{v}$ dos vectores de dimensión n definidos sobre el mismo sistema de referencia:

Distancia Manhattan o distancia L1

$$d_{L1} = ||\vec{v} - \vec{u}||_{L1} = \sum_{i=1}^{n} |v_i - u_i| \tag{3.1}$$

Distancia Euclídea o distancia L2

$$d_{L2} = ||\vec{v} - \vec{u}||_{L1} = \sqrt{\sum_{i=1}^{n} (v_i - u_i)^2} \tag{3.2}$$

Distancia *X*-Cuadrado

$$d_{X^2} = ||\vec{v} - \vec{u}||_{X^2} = \sum_{i=1}^{n} \frac{(v_i - u_i)^2}{v_i + u_i} \tag{3.3}$$

Distancia HIK: *Histogram Intersection Kernel*

$$d_{X^2} = ||\vec{v} - \vec{u}||_{X^2} = \sum_{i=1}^{n} \text{mín}\{v_i, u_i\} \tag{3.4}$$

3.1.5. Normal de una superficie

El vector normal $\vec{n}$ en un punto $p_q \in P$, siendo P una nube de puntos es el vector perpendicular a la superficie en el punto p_q. En el cálculo del vector normal se utiliza un entorno de vecindad P^r. Por lo tanto, el tamaño del radio y el número de puntos de P^r influyen en el cálculo de $\vec{n}$.

Una forma sencilla de estimar $\vec{n}$ es calculando primero el plano tangente a P^r en el punto p_q. Este plano se define como,

$$\prod_q(x,y,z) = a(x-x_q)+b(y-y_q)+c(z-z_q) = 0 \tag{3.5}$$

siendo el vector $\vec{n} = \{a,b,c\}$.

Para estimar el plano que mejor se aproxima a P^r hay que reducir la distancia entre $\prod_q$ y cada $p_i \in P^r$. De esta manera el problema se resuelve con mínimos cuadrados [2] del sistema obteniendo los valores *a, b* y *c* que minimicen la ecuación 3.6

$$f(a,b,c) = \sum_{i=1}^{n}(a(x_i-x_q)+b(y_i-y_q)+c(z_i-z_q))^2 \tag{3.6}$$

Existe una ambigüedad en la dirección de $\vec{n}$, ya que este puede tener una dirección hacia el centro o el exterior del objeto. Para resolver esta ambigüedad se necesita saber el punto de vista con el que fue adquirida la nube de puntos.

Otra forma de calcular $\vec{n}$ en p_q es realizando un análisis de los valores y vectores de la matriz de covarianza C_q. Esta se define aplicando la técnica de Análisis de Componentes Principales [3] del siguiente modo.

Primero se calcula la matriz 3x3 de covarianza C_q:

$$C_q = PP^T = \begin{bmatrix} p_1-\bar{p} \\ \dots \\ p_n-\bar{p} \end{bmatrix} \begin{bmatrix} p_1-\bar{p} \\ \dots \\ p_n-\bar{p} \end{bmatrix}^T \tag{3.7}$$

Siendo $\bar{p}$ el centroide de P^r y n el número de puntos que lo componen. Utilizando el método de Descomposición en Valores Singulares (SVD) [4] se obtienen los autovalores λ_j y autovectores $\vec{v}_j$ de la matriz C_q:

$$C_q \cdot \vec{v}_j = \lambda_j \cdot \vec{v}_j \tag{3.8}$$

El autovector de menor autovalor, $\vec{v}_0$, es una aproximación del vector $\vec{n}$ normal a

P^r en p_q, ya que esa es la dimensión degenerada en una nube de puntos plana.

3.1.6. Curvatura de una superficie

El cálculo de la curvatura c en un punto $p_q \in P$, siendo P una nube de puntos, es la variación existente en las normales de los puntos $p_i \in P^r$ siendo P^r el entorno de vecindad del punto p_q. Al igual que con el cálculo de la normal, aquí influye de manera significativa el tamaño del radio y el número de puntos que se ha utilizado para el cálculo de P^r.

Para el cálculo de la curvatura se utiliza el método de Pauly [5], que es invariante a la escala, donde la curvatura puede ser calculada de la siguiente forma:

$$c_q = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{3.9}$$

Siendo λ_0, λ_1 y λ_2 los valores propios de la matriz de covarianza C_q, ecuación 3.7, del cálculo del vector normal $\vec{n}$. Ver sección 3.1.5.

3.1.7. Marco de Darboux de una superficie

El marco de Darboux es una herramienta que permite calcular la relación existente entre cada punto de la nube y sus vecinos. De este modo, permite representar matemáticamente la curvatura de la superficie con tres vectores. Estos tres vectores, según Darboux, son el vector normal $\vec{n}$, el vector tangente $\vec{t}$, y el producto vectorial de estos.

Para el cálculo del marco de Darboux en un punto p_q, se parte de un entorno de vecindad P^r y se considera una curva sobre p_q, el marco se define como $\{\vec{u}, \vec{v}, \vec{w}\} = \{\vec{n}_q, \vec{t}_q, \vec{n}_q \times \vec{t}_q\}$, donde $\vec{n}_q$ es el vector normal unitario en el punto p_q y $\vec{t}_q$ es el vector tangente en el punto p_q.

3.2. Tipos de Descriptores de superficie

A veces una nube de puntos puede representar un objeto 3D aislado. En procesos de reconocimiento de objetos es común crear descriptores como conjuntos de propiedades que lo describan. Son estos descriptores los que permiten realizar procesos de comparación entre diferentes objetos, realizando un análisis de semejanzas de las propiedades de los objetos a comparar. En este apartado se explicarán algunos de los descriptores más comunes. Los descriptores que se abordan son descriptores obtenidos a partir de nubes de puntos y estos se pueden clasificar en tres grupos:

- Basados en propiedades de puntos de la nube de puntos y su entorno de vecindad.
- Basados en propiedades de punto de vista.
- Basados en la orientación de la superficie.

Los siguientes descriptores hacen referencia a los conceptos explicados en los apartados: (3.1.2), (3.1.3), (3.1.4), (3.1.5), (3.1.6) y (3.1.7).

Para más información sobre descriptores 3D ver [6]

3.2.1. PFH: *Point Feature Histogram*

Este descriptor se conoce como Histograma de Características de Punto y es uno de los descriptores más importantes. Este es la base de otros descriptores como FPFH. El algoritmo de PFH calcula la tupla $(\alpha, \Phi, \theta, d_{L2})$ para cualquier par de puntos p_i y p_j pertenecientes a P^{r2}, siendo P^{r2} el conjunto de vecinos del entorno de vecindad de doble radio. Ver apartado 3.1.3.

Para calcular la variación geométrica entre p_j y p_i, siendo estos pertenecientes a P^{r2}, separados a una distancia euclidiana d_{L2} (Ver apartado 3.1.4), partiendo de que ya se han calculado todos los vectores normales de los puntos pertenecientes a P^{r2} y se ha definido un sistema de referencia fijo (Marco de Darboux) en uno de los puntos, por ejemplo p_j. Así este marco queda definido por $\{\vec{u}, \vec{v}, \vec{w}\} = \{\vec{n}_j, \vec{t}_j, \vec{n}_j \times \vec{t}_j, \}$

obtenemos que:

$$\vec{u} = \vec{n}_j, \;\; \vec{v} = \vec{t}_j = \vec{u} \times \frac{(p_i - p_j)}{d_{L2}}, \;\; \vec{w} = \vec{u} \times \vec{v} = \vec{n}_j \times \vec{t}_j \tag{3.10}$$

Entonces la variación geométrica entre los puntos p_j y p_i se puede expresar como la diferencia relativa entre la dirección de los vectores normales $\vec{n}_j$ y $\vec{n}_i$ de tal manera que:

$$\alpha = \arccos(\vec{v} \cdot \vec{n}_i), \;\; \phi = \arccos(\vec{u} \cdot \frac{(p_i - p_j)}{d_{L2}}), \;\; \theta = \arctan(\vec{w} \cdot \vec{n}_i, \vec{u} \cdot \vec{n}_i) \tag{3.11}$$

Este tipo de descriptor tiene 6 grados de libertad, 3 de traslación y 3 de rotación. En general, los descriptores representan el número de veces que se repite una tupla, en este caso (α, Φ, θ). Para ello es necesario crear subdivisiones de cada elemento de la tupla, representando cada una de estas subdivisiones un rango de valores. Para que exista un correlación entre los valores de la tupla, es necesario crear el mismo número de subdivisiones para cada elemento de la tupla. En este caso dado que los 3 valores son angulares, el criterio utilizado es la máxima diferencia angular, para determinar el máximo número de subdivisiones. En general PFH trabaja con 5 subdivisiones obteniendo un máximo de 125 combinaciones.

El principal problema de este descriptor es el coste computacional. Para que PFH sea más eficiente se puede despreciar la componente d, obteniéndose una tupla del tipo (α, Φ, θ).

3.2.2. FPFH: *Fast Point Feature Histogram*

Este descriptor está basado en el PFH (Ver sección 3.2.1) y solventa el problema del coste computacional. El cálculo de este descriptor se realiza en dos pasos:

1. Para cada punto candidato p_q con P^{r2} se calcula el conjunto de tuplas (α, Φ, θ) de p_q a $p_i \in P^{r2}$ con la ecuación 3.11. El histograma resultante se denomina SPFH (*Simply Point Feature Histogram*).

2. Para cada $p_i \in P^r$ se considera su propio entorno de vecindad P_i^{r2} y se calcula el SPFH ponderando los valores con unos pesos según la distancia de p_q a p_i.

Entonces obtenemos como resultado que:

$$FPFH(p_q) = SPFH(p_q) + \frac{1}{k}\sum_{i=1}^{k}(\frac{i}{\omega_i}SPFH(p_i)) \tag{3.12}$$

FPFH trabaja con 11 subdivisiones que darían como resultado 1331 combinaciones posibles, pero al tratarse de 3 histogramas independientes concatenados el número de combinaciones se reduce a 33.

El descriptor PFH se comporta mejor que FPFH frente a la presencia de ruido y con entornos de vecindad donde varia la densidad.

3.2.3. VFH: *View Feature Histogram*

Este tipo de descriptores está basado en el descriptor FPFH (Ver sección 3.2.2), ya que este es robusto al ruido y a la escala pero no al punto de vista con el que se capturó la escena. VFH contiene la misma información que FPFH, añadiendo información para codificar la información del punto de vista. Por lo tanto, puede emplearse VFH para procesos de reconocimiento y cálculo de la pose.

VFH tiene un descriptor de punto de vista y un descriptor de la forma de la superficie.

Para calcular el descriptor de punto de vista en un punto, primero se calcula el centroide de la superficie p_c fijando en este un sistema de referencia, cogiendo la dirección de referencia como el vector normal a la superficie $\vec{n}_c$ en el punto p_c. De este modo se codifica el descriptor como la desviación existente entre $\vec{n}_c$ y $\vec{n}_j$ de cada $p_j \in P$, de manera similar a como se hace en PFH y FPFH (Ver secciones 3.2.1 y 3.2.2). Asumiendo $p_j = p_c$ en la ecuación 3.10 para así definir el marco de Darboux, midiendo posteriormente la variación angular con la ecuación 3.11 entre el punto de referencia p_c y los puntos p_j de la superficie de la nube de puntos, siendo $\vec{n}_i = \vec{n}_j$.

Algunos de los problemas que tiene este descriptor son:

- No permite calcular la posición de objetos cuando existen giros sobre el eje del punto de vista de la cámara (eje z), generando duda si se desea estimar la *pose* del objeto.

- Presenta deficiencias en reconocimiento de objetos dado que es sensible a pérdidas de información.

Para más información ver [7].

3.2.4. CVFH: *Clustered View Feature Histogram*

El descriptor CVFH fue creado como extensión de VFH (Ver sección 3.2.3) para solventar alguno de los problemas que tiene VFH. La principal ventaja de CVFH frente a VFH es que CVFH no tiene en cuenta aquellas partes de la nube que no se han capturado de forma robusta, como son las aristas.

El algoritmo de CVFH consiste en dividir la nube de puntos P en regiones estables *"Clusters"* $P = P_1, P_2, ..., P_n$. Cada subconjunto P_k representa una parte del objeto, cuyos puntos contienen poco ruido.

Para calcular el descriptor CVFH de una nube de puntos P, primero ha de realizarse el cálculo de los *"clusters"*, para ello se comienza eliminando aquellos puntos que tienen un alto nivel de curvatura. Estos son puntos atípicos, debido a que están próximos a aristas o porque se han capturado con ruido. A continuación, se agrupan los puntos p_i en función de la dirección de los vectores normales en estos puntos. Un conjunto P_k está compuesto por puntos p_{ki} cuya variación en dirección de todos los puntos p_{ki} sea mínima y la variación entre los valores medios de la dirección del vector normal de todos los *clusters* $P_1, P_2, ..., P_n$ sea máxima.

Una vez son conocidas las P_k regiones estables, se calculan los descriptores VFH para cada una. Primero, se define el punto de vista de la región estimando el centroide p_{kc} y la normal $\vec{n}_{kc}$. Y después, se calculan (α, Φ, θ) para cada punto $p_{ki} \in P_k$ con respecto a p_{kc}.

Para más información ver [8].

3.2.5. OUR-CVFH: *Oriented, Unique and Repeatable - Clustered View Feature Histogram*

El descriptor OUR-CVFH fue creado como extensión de CVFH (Ver sección 3.2.4), para solventar alguno de las deficiencias de CVFH.

OUR-CVFH se basa en el uso de SGURF *Semi-Global Unique Reference Frames*, que consiste en calcular sistemas de coordenadas irrepetibles para cada región. Primero se descompone la nube de puntos P en P_k regiones, de manera similar a como se realiza en CVFH. Luego se añade una nueva etapa de filtrado, eliminando de las regiones los puntos p_{ki} cuyo ángulo entre $\vec{n}_k$ y $\vec{n}_i$ supere un umbral. De esta manera se reduce el tamaño de las regiones.

Finalmente, para cada región P_k se calcula el descriptor CVFH utilizando el sistema de referencia obtenido con SGURF.

Se pueden ver más detalles de este método en [9].

3.3. Diferentes técnicas de ajuste de nubes de puntos a modelos

En la actualidad existen diferentes técnicas de ajuste de nubes de puntos. A continuación, se explican dos de las más utilizadas.

3.3.1. RANSAC-Model: *RANdom SAmple Consensus - Model*

RANSAC [10] es un método iterativo utilizado para calcular los parámetros de un modelo matemático de un conjunto de datos que contienen valores atípicos. Se trata de un algoritmo no determinista, dado que produce un resultado que tiene una alta probabilidad de ser el correcto. Al aumentar el número de iteraciones esta probabilidad mejora. RANSAC tiene como entrada un conjunto de valores de los datos observados (nube de puntos). El modelo al que debe realizarse el ajuste en el proceso de fitting se tratará de una nube de puntos. El algoritmo logra su objetivo mediante la repetición de

los siguientes pasos:

Seleccionar un subconjunto aleatorio de los datos originales que serán los *"inliers"* hipotéticos.

Crear un modelo con el conjunto de estos *"inliers"*.

while iteraciones $\leq$ máximo iteraciones AND conjunto consenso no tiene suficientes puntos **do**

for all $p_i \in P$ **do**

if distancia entre p_i y el modelo ajustado $\leq$ umbral **then**

Añadir p_i al conjunto consenso

end if

end for

end while

Este proceso se repite un número finito de veces, obteniéndose un modelo que dependiendo de si tiene suficientes puntos en el conjunto de consenso, puede ser aceptado o rechazado.

Para más información ver [11].

3.3.2. ICP: *Iterative Closest Point*

ICP es un algoritmo utilizado para ajustar dos nubes de puntos. ICP se utiliza usualmente para reconstruir superficies 2D o 3D a partir de diferentes análisis, para localizar robots y lograr planificar trayectorias óptimas (cuando la odometría no es fiable debido al terreno resbaladizo).

ICP tiene como entrada dos nubes de puntos, destino P^d y fuente P^f. P^d se mantiene estática mientras que P^f se va transformando para adaptarse mejor a ella. Para adaptarse cada vez mejor, el algoritmo revisa iterativamente la transformación de tal manera que se minimice un error métrico que es por lo general la distancia euclidiana entre P^f y P^d.

ICP logra su objetivo mediante la iteración de los siguientes pasos:

for Error $\leq$ umbral AND iteraciones $\leq$ máximo de iteraciones **do**

for all $p_i \in P^f$ **do**

Obtener $p_j \in P^d$ cuya distancia entre p_j y p_i sea menor.

end for

Se estima la rotación y traslación mediante la técnica de mínimos cuadrados que minimiza la distancia anterior.

Transformar P^f mediante el resultado obtenido en el paso anterior.

end for

Este proceso se repite un número finito de veces o hasta que se ha minimizado el error hasta un cierto umbral, obteniéndose un modelo que dependiendo de si tiene suficientes puntos en el conjunto de consenso, puede ser aceptado o rechazado.

Para más información ver [12].

3.4. Extracción de objetos de una nube de puntos

Para la realización de este proceso existe el algoritmo llamado *"Euclidean Cluster Extraction"* que realiza la separación de los distintos objetos de una escena mediante la métrica de la distancia euclidiana (Ver 3.1.4).

El algoritmo realiza los siguientes pasos:

Creación del *Kd-tree* a partir de la nube de puntos P.

Establecer una lista de agrupaciones vacía C y una lista de puntos que necesitan ser comprobados Q.

for all $p_i \in P$ **do**

Añadir p_i a Q

for all $p_i \in Q$ **do**

Buscar el conjunto de puntos de vecinos P^r del punto p_i en una esfera de radio $r < d_th$ siendo d_th la distancia umbral para separar objetos.

Para cada $p_j \in P^r$ si no ha sido procesado añadirlo a Q.

end for

Cuando todos los puntos de Q se hayan procesado, añadir Q a la lista de *Clusters*

C , y vaciar Q.

end for

El algoritmo termina cuando todos los puntos de P han sido procesados.

Se puede encontrar más información sobre el funcionamiento del algoritmo en [13]

3.5. El robot Shelly

Shelly es el robot asistencial más avanzado con el que cuenta RoboLab actualmente. Su desarrollo está soportado por varios proyectos de investigación, incluyendo el proyecto del Plan Nacional de Investigación, en su convocatoria Retos de la Sociedad, TIN2015-65686-C5-5-R de nombre *Fusión de las habilidades de navegación y manipulación para robots sociales en SmartHomes*, por el plan de ayuda a grupos de investigación de la Junta de Extremadura, GR15120, por la Red de Excelencia del Ministerio de Economía y Competitividad, Red de Agentes Físicos TIN2015-71693-REDT, por el proyecto de colaboración internacional España-Brasil del Ministerio de Educación, Cultura y Deporte PHBP14/00083 y por el proyecto europeo de la convocatoria POPTEC, EuroAGE. El objetivo de este robot es extender la autonomía de las personas mayores o de personas con dependencias para permitirles llevar una vida independiente en sus hogares. Actualmente, el robot se encuentra en un laboratorio diseñado específicamente para ello y que emula un apartamento habitado.

Figura 3.5: Imagen de Shelly

A continuación, se procede a realizar una descripción de los componentes físicos de los que dispone Shelly.

Como se puede observar en la figura 3.5, Shelly dispone de unas ruedas Mecanum que le permiten moverse en cualquier dirección o girar sobre si mismo; dispone de un LIDAR para saber la distancia a la que se encuentra de los obstáculos cercanos; un brazo robótico central con el que puede coger los diferentes objetos; una cámara ASUS Xtion Pro que proporciona imágenes RGBD; una cámara Kinect 2 para la localización

y seguimiento de las personas en su entorno; 4 mini-ordenadores NUC conectados a través de un conmutador sobre los que se ejecutan los componentes; una pantalla táctil, baterías y la electrónica de potencia y recarga.

Actualmente, está dotado con un sistema de reconocimiento y localización de marcas especiales, AprilTags, que se pegan a los objetos en su parte superior, como el que se puede observar en la figura 3.6a. Este sistema es el que se desea reemplazar con el sistema de reconocimiento y obtención de pose de objetos sin marcas externas. El sistema final deberá poder reconocer y calcular la pose de objetos como el de la figura 3.6b.

(a) Envase de fideos con marca (b) Envase de fideos sin marca

Figura 3.6: (3.6a Objeto reconocible por el sistema de reconocimiento de objetos inicial) (3.6b Objeto no reconocible por el sistema de reconocimiento de objetos inicial)

3.6. RoboComp y CORTEX

Antes de explicar qué es el *framework* de desarrollo para robótica, RoboComp, se hará una breve explicación de qué es la programación orientada a componentes (COP).

La programación orientada a componentes es una técnica utilizada para la creación de software que trata de solventar dos de los problemas principales de este campo: la

escalabilidad y la **reutilización**.

Un componente es un programa software, que forma parte de un sistema software mayor. Este ofrece unos servicios a través de una interfaz predefinida. Las características principales de un componente son las siguientes:

1. Ser reutilizable.
2. Ser intercambiable.
3. Poseer interfaces definidas.
4. Ser cohesivos.

En el caso de RoboComp al usar *ice* lo único que se necesita para poder realizar una conexión de un componente a otro es:

<Nombre de la Interfaz> : <TCP / UDP> -p puerto -h host

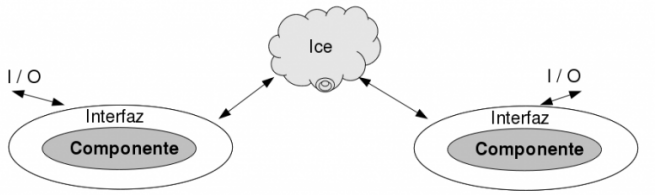

Figura 3.7: Representación genérica de los componentes

Una vez se conoce que es la programación orientada a componentes se procede a la explicación de qué es RoboComp.

RoboComp [14] es un *framework* de desarrollo para robótica de código abierto que ofrece la posibilidad de crear componentes de una manera fácil y sencilla, comunicándose éstos a través de interfaces públicas. Las comunicaciones se realizan a través del *middleware Ice*. Para generar un componente con RoboComp se utilizan un lenguaje de domino específico, un ejemplo de este lenguaje sería el siguiente:

```
import ”import1.idsl”;
import ”import2.idsl”;
```

```
Component ComponenteName
{
  Communications
  {
    implements interfaceName;
    requires otherName;
    subscribesTo topicToSubscribeTo;
    publishes topicToPublish;
  };
  language Cpp/Python;
  gui Qt(QWidget);
};
```

De esta manera un componente generado con RoboComp puede subscribirse a un tópico y/o publicar tópico y/o implementar una interfaz y/o hacer uso de una interfaz, pudiendo usarse como lenguajes de programación *Python* o *C++*. Los componentes generados con RoboComp tienen un archivo de configuración en el que configurar el puerto y la ip de una interfaz implementada en otro componente, el tamaño de los mensajes de Ice, añadir variables de configuración del componente, etc. Además RoboComp cuenta con otro lenguaje específico de domino para la definición de las interfaces, estos archivos son los *.idsl*, similares a las interfaces de *Ice*. Gracias al *framework* de *Ice* los componentes escritos con RoboComp pueden comunicarse con componentes escritos en otros lenguajes, como Java, PHP, JavaScript, etc. Además RoboComp utiliza otras herramientas como *CMake, Qt4, Qt5, IPP, OpenSceneGraph* y *OpenGL*. RoboComp también cuenta con un simulador.

El objetivo de RoboComp es lograr una mayor eficiencia, simplicidad y capacidad de reutilización de esos componentes.

Para más información ver [15]

Todos los componentes que controlan a Shelly han sido creados utilizando RoboComp.

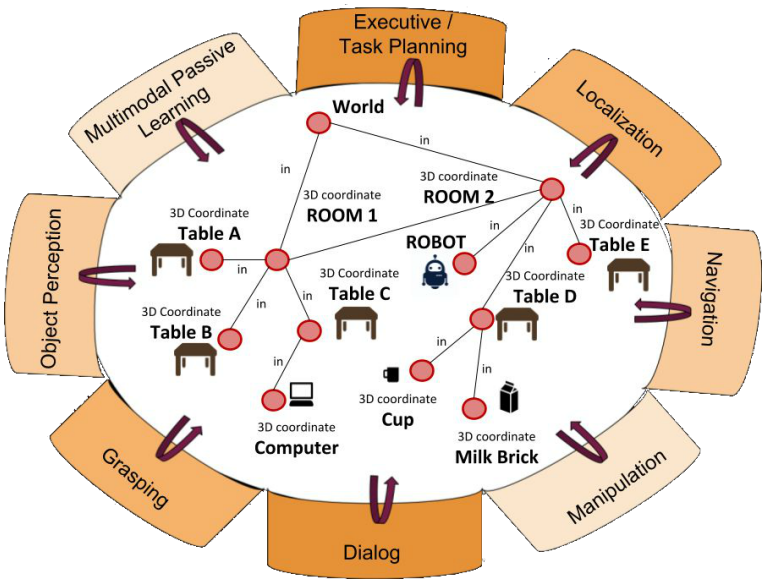

Figura 3.8: Representación de la arquitectura cognitiva CORTEX como un conjunto de agentes que comparten una representación común.

Utilizando RoboComp como base de desarrollo, RoboLab en colaboración con otras universidades ha creado una arquitectura cognitiva para robots denominada *CORTEX* [16]. La figura 3.8 muestra la estructura básica de esta propuesta. Esta arquitectura está diseñada siguiendo las siguientes pautas:

- Está compuesta por una colección de agentes que interactúan entre sí.
- Cada agente está internamente compuesto por una red de componentes creados con RoboComp.
- Los agentes pueden estar en cualquier parte del espectro deliberativo-reactivo.
- La creencia global del robot sobre sí mismo y su entorno está codificada en una estructura gráfica, denominada Representación Profunda del Estado (DSR), que es compartida entre los agentes.
- DSR es un gráfico híbrido que codifica el conocimiento cinemático y simbólico. Los nodos en DSR son atributos simbólicos que pueden extenderse con propiedades métricas. Los enlaces codifican predicados de lógica binaria.

- Todos los conocimientos que entran en el sistema, es decir, el aprendizaje del espacio, están restringidos por la estructura del DSR. Cada pieza de información se coloca en relación con otras piezas existentes, siendo estos enlaces los pilares de la interpretación semántica.
- DSR codifica el conocimiento en diferentes niveles de abstracción, desde datos en bruto en las hojas hasta conceptos abstractos en niveles superiores. Los agentes elaboran datos de fila en información útil y conocimiento utilizable leyendo el grafo, procesando y escribiendo de nuevo en él.
- DSR proporciona un contexto dinámico a los cálculos locales que tienen lugar dentro de los agentes.
- Los agentes se comunican entre ellos sólo a través de cambios en el gráfico DSR.
- Hay agentes especiales que, usando la tecnología de simulador de RoboComp, pueden emular cursos de acciones futuras partiendo del estado actual en DSR. Estos agentes se llaman emuladores y proporcionan una visión del futuro a diferentes niveles de abstracción que puede compararse con el despliegue real, mensurable de eventos para derivar un error de predicción. Este error va a ser utilizado por los otros agentes para mejorar sus habilidades de planificación, aprendizaje, control o calibración.

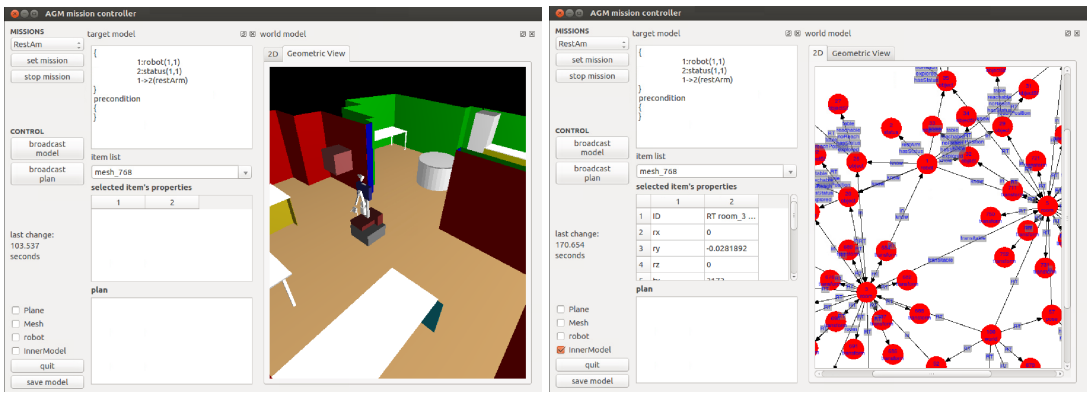

Figura 3.9: a) Visualización gráfica del DSR de CORTEX durante una ejecución; b) Visualización en forma de grafo del DSR de CORTEX durante la misma ejecución.

Capítulo 4

Un sistema de reconocimiento de objetos para su manipulación por robots

Una vez llegado este punto ya se ha descrito la situación actual de Shelly (físicamente y la estructura interna), y qué es y para qué sirven RoboComp y CORTEX. Se continúa ahora con la explicación del sistema de reconocimiento de objetos y obtención de su pose.

Dado que en Robolab se ha trabajado con anterioridad con cámaras 3D, ya se dispone de los componentes *openni2RGBD* y *openni2PCL* que ofrecen a través de una interfaz, la nube de puntos de la escena capturada en el sistema de referencia de la cámara. Para trabajar con las nubes de puntos capturadas a través del componente anterior se hace uso de la librería *Point Cloud Library (PCL)*. En esta Librería se pueden encontrar una gran variedad de algoritmos para el tratamiento de las nubes de puntos (p.e. *Random Sample Consensus* o RANSAC, *VoxelGrid*, *Iterative Closest Point* o ICP).

Para describir de una forma más precisa el proceso que sigue el sistema, éste se dividirá en subprocesos:

4.1. Entrenamiento del sistema de reconocimiento de objetos

La primera tarea a realizar en un sistema de reconocimiento de objetos es su entrenamiento. Para el entrenamiento del sistema se decidió utilizar un método supervisado. La fase de entrenamiento requiere obtener los descriptores de los objetos que posteriormente se desea reconocer, pero para obtenerlos hay que realizar un procesamiento previo de la nube de puntos capturada.

4.1.1. Cambio del sistema de referencia de la escena

Una vez disponemos de la nube de puntos de la escena, el primer paso es cambiar el sistema de referencia a la base del robot para que el eje Y sea perpendicular al suelo. Para ello, se hace uso de la clase InnerModel de RoboComp. InnerModel puede contener la descripción física del entorno de una habitación, el cuerpo de un robot, etc. Del InnerModel del robot de puede obtener la transformación que permite representar un punto del sistema de referencia de la cámara en el sistema de referencia del robot $M_{T_{camaraabase}}$, situado en el suelo. Para representar la nube de puntos de la escena en el sistema de referencia a la base del robot, se multiplica cada uno de los puntos por la matriz de transformación $M_{T_{camaraabase}}$.

En las figuras 4.1a y 4.1b se muestra un ejemplo de la nube de puntos de la escena.

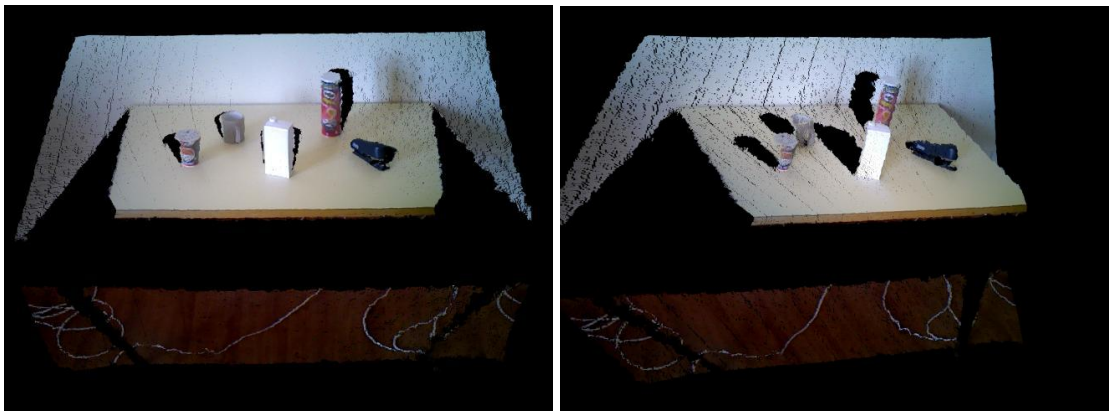

Figura 4.1: 4.1a Nube de puntos de la escena de frente. 4.1b Nube de puntos de la escena rotada.

4.1.2. Segmentación de objetos

Partiendo de la suposición de que los objetos siempre estarán sobre una superficie plana suficientemente grande que ocupa la mayor parte de la nube de puntos (ej. mesa), se calcula una función que represente al plano usando el algoritmo **RANSAC** explicado en la sección 3.3.1. Una vez obtenida la ecuación del plano, éste se elimina de la escena, incluidos los puntos que se encuentren debajo de él y dejando únicamente los que estén por encima de este plano como puntos pertenecientes a objetos.

En las figuras 4.2a y 4.2b se muestra el plano de la mesa obtenido tras procesar la nube de puntos de la escena mostrada en las figuras 4.1a y 4.1b.

Figura 4.2: 4.2a Nube de puntos de la plano de frente. 4.2b Nube de puntos de la plano rotada.

Una vez el sistema dispone de la nube de puntos que contiene a los objetos, se llevará a cabo una agrupación euclidiana con un umbral de 2 cm de distancia. Para realizar este proceso se utiliza el algoritmo *"Euclidean Cluster Extraction"* explicado en la sección 3.4. Como resultado se obtiene una lista en los que cada elemento representa la nube de puntos de un objeto de la escena.

En las figuras 4.3a y 4.3b se puede ver la nube de puntos que este algoritmo tendría como entrada.

Figura 4.3: 4.3a Nube de puntos de los objetos de frente. 4.3b Nube de puntos de los objetos rotada.

4.1.3. Extracción de descriptores de objetos

Una vez se tiene la escena segmentada con las nubes de puntos de los distintos objetos, se pasa a extraer de éstas las características visuales para posteriormente realizar el emparejamiento. Con la librería *PCL* se pueden obtener distintos tipos de descriptores atendiendo cada uno de estos a diferentes características de la nube de puntos (p.e. forma, curvatura,...). Concretamente, en este sistema se ha optado por la utilización de los descriptores **VFH, CVFH** y **OUR-CVFH**, que ya se abordaron en los apartados 3.2.3, 3.2.4 y 3.2.5.

Para que el usuario pueda elegir entre los tres tipos de descriptores, los componentes creados con RoboComp tienen un archivo de configuración, en el cual se pueden añadir variables que pasarán al componente al iniciarse. En este fichero se ha añadido una variable que puede tomar los valores (**VFH, CVFH, OUR-CVFH**), haciendo que sea más fácil y sencilla la transición de un tipo de descriptor a otro.

4.2. Información a almacenar

Llegado este punto se plantea la siguiente pregunta **¿qué se debe guardar de cada objeto?** y **¿cómo se debe guardar?** Dado que el sistema final debe también obtener la posición de los objetos, la información a guardar será: **etiqueta del objeto**, **la nube de puntos de la vista de un objeto**, **los descriptores** y **la pose del objeto en cada vista**, Además para mejoras futuras se almacenará también la **imagen de la vista**.

Hasta ahora el sistema tiene toda esa información a excepción de la pose del objeto en la vista de entrenamiento. A continuación, se explicará el proceso seguido para el cálculo de esta pose.

4.2.1. Cálculo de la pose de un objeto en la vista de entrenamiento

La nube de puntos de un objeto representa únicamente la superficie visible de éste desde el punto de vista de la cámara, con lo cual el cálculo de la pose de la vista de entrenamiento debe realizarse de manera externa a las nubes de puntos ya que si, por

ejemplo, para un objeto que no sea un prisma de revolución tomamos como pose de la vista el centroide de la nube, para otra vista de ese mismo objeto podemos obtener una pose totalmente diferente.

Con el fin de solventar este problema se imprimió un plano con 9 códigos QR (tag), y se tomaría como pose del objeto el *tag* central. De tal manera que cuando se visualiza en la imagen cualquier otro tag, el sistema puede calcular la pose del tag central mediante matrices de transformación. Estas matrices de transformación están guardadas en un fichero XML, para que sí, en un futuro el mapa cambiase, solo fuese necesario cambiar este archivo XML. Este archivo se puede leer con la clase Innermodel, pudiendo obtener de este las transformaciones de un *tag* a otro.

En la figura 4.4 podemos visualizar las imágenes tomadas para el entrenamiento del sistema con una Taza, en las que aparece el objeto sobre el *tag* central del plano comentado anteriormente.

La pose de un *tag* que está visible a través de una cámara, la publica un componente existente llamado *AprilTag*, utilizado en proyectos anteriores por RoboLab.

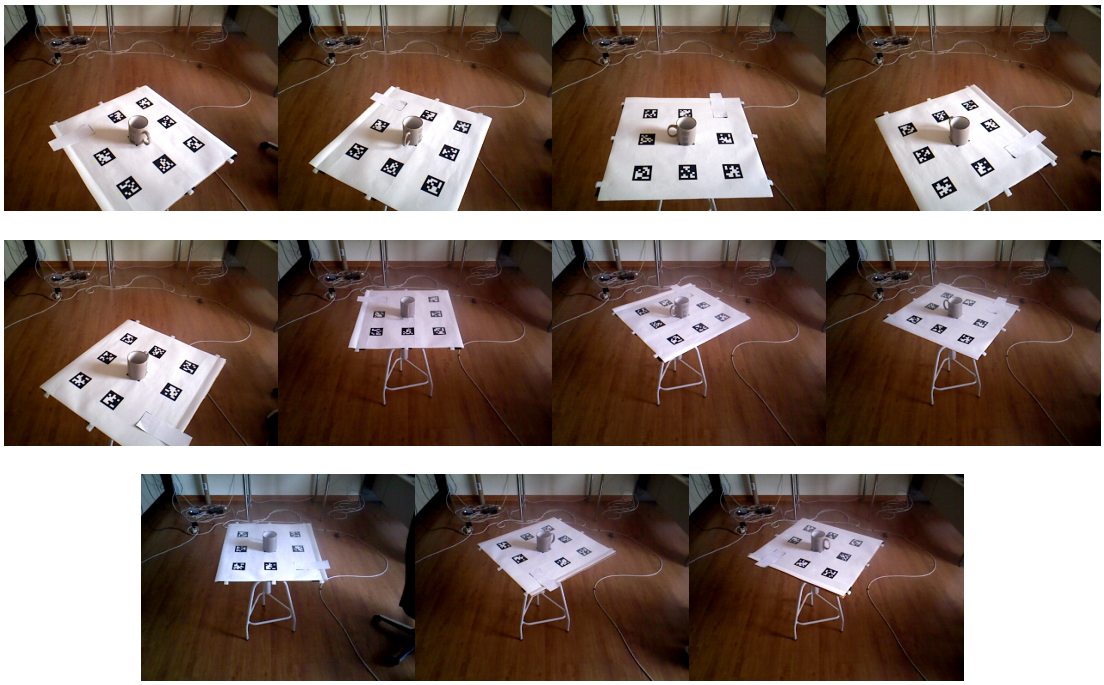

Figura 4.4: Ejemplo entrenamiento Taza de leche

4.2.2. Almacenamiento de la información

Para el almacenamiento de la información obtenida del entrenamiento se decidió crear un árbol de directorios en el que toda la información de un objeto estaría almacenada en un directorio con la etiqueta de dicho objeto.

Dentro de este directorio estarían almacenados las nubes de puntos del objeto, las imágenes del objeto, y los descriptores de dichas nubes. Además, la pose de las distintas vistas del objeto, se almacenarían en un fichero .xml, con líneas como el siguiente formato:

<transform id = "pose_0_etiqueta" tx = "56.2" ty="715.8" tz="831.5" rx="1.527" ry="0.00918" rz="0.0288" >

4.2.3. Carga de descriptores desde la información almacenada

Una vez iniciado el componente, se realiza una búsqueda de las nubes de puntos en un directorio y por cada nube encontrada, si no existe su descriptor lo genera, lo guarda y lo añade al buscador de correspondencias (*matcher*), almacenando junto a él la ruta de la nube de puntos (*path*), y la etiqueta, que sería el nombre de la carpeta contenedora. La lista de descriptores se conoce como formación (*training*).

4.3. Corrección de la pose calculada en el entrenamiento

Se observó que en ciertas capturas la pose calculada con las marcas no correspondía con la pose deseada. Con el fin de solventar este problema se añadieron tres *sliders* (x,y,z) para poder corregir de forma manual y seleccionar la pose deseada. Esta corrección se puede visualizar en tiempo real, a través de una ventana de PCL que permite mostrar la nube de puntos.

Además, para poder modificar la pose de capturas ya guardadas en otros entrenamientos, se decidió crear otro componente que ofreciese este servicio.

4.4. Reconocimiento de objetos

Una vez el sistema está entrenado y se tiene toda la información necesaria almacenada, éste puede reconocer los diferentes objetos añadidos en el entrenamiento.

Para reconocer los objetos de una escena se realizarán los siguientes subprocesos:

- Segmentación de objetos (Explicado en el apartado 4.1.2).
- Extracción de descriptores de objetos (Explicado en el apartado 4.1.3).
- Obtención de las semejanzas y asignación de la etiqueta.

Cuando el sistema ya tiene la información producida por los subprocesos de segmentación de objetos y extracción de descriptores de objetos, se puede iniciar el proceso de obtención de las semejanzas y asignación de la etiqueta.

4.4.1. Obtención de las semejanzas (*Matching*) y asignación de la etiqueta

Dado que ya se dispone de los descriptores de cada objeto de la escena, para cada descriptor se realiza la comparación de este con los descriptores del entrenamiento (*training*). De esta comparación se obtendrá una distancia entre el descriptor de un objeto y cada uno de los descriptores del entrenamiento, esta distancia representa la razón de semejanza entre ambos descriptores. El descriptor del entrenamiento que obtenga la menor distancia sera el ganador, y se asignará a ese objeto la etiqueta del descriptor ganador.

Pero es posible que dos descriptores del entrenamiento tengan distancias muy similares, con lo cual para descartar falsos positivos comparamos la menor distancia y la siguiente menor distancia con etiqueta diferente. Si la distancia entre ellas es pequeña, se dice que se encuentra con el caso de un falso positivo, de lo contrario se dirá que el objeto de la escena es del tipo del descriptor de menor distancia.

Por ejemplo, si en una mesa hay un objeto y se obtiene el siguiente resultado al realizar el *matching*:

- **brick_pose1 = 3040** —— Primera menor distancia d_1
- brick_pose3 = 3050
- **pringles_pose3 = 3500** —— Segunda menor distancia con etiqueta diferente d_2
- pringles_pose4 = 3800
- brick_pose2 = 4000
- ...

Si la diferencia entre la primera y la segunda distancia es lo suficientemente grande se aceptará el resultado. Para ello se realiza la normalización de las distancias: d_1/d_2 y si el resultado supera un umbral por ejemplo del 85 % se dice que se desconoce el objeto de la escena. En este caso el resultado es de 86.857 %, como supera dicho umbral se considera como un objeto desconocido. En caso de que el resultado de esta división sea inferior al umbral, se le asignará la etiqueta del descriptor ganador, que en este caso sería *brick*.

El proceso de *matching* se realiza de la misma forma independientemente del tipo de descriptor que se haya seleccionado (**VFH, CVFH** o **OUR-CVFH**).

4.5. Obtención de la información que devuelve el sistema de un objeto

Un buen sistema de reconocimiento y localización de objetos debe devolver la información relevante de los objetos. Por ello, se considera que de cada objeto de la escena se devolverían los siguientes datos:

- Etiqueta del objeto (Solo si se ha logrado reconocer).
- Pose del objeto (Solo si se ha logrado reconocer).
- Cubo delimitador o *Bounding-box*.

A continuación, se explican los procesos seguidos para el cálculo de estos datos.

4.5.1. Ajuste de nubes de puntos

Cuando el sistema ya ha designado una etiqueta a un objeto de la escena, el sistema conoce el *path* de la nube ganadora. Entre la nube del objeto de la escena y la nube ganadora se debe realizar un ajuste, para posteriormente calcular la pose del objeto.

El ajuste de nubes de puntos consiste en unir dos nubes de puntos en otra rígida, reduciendo al mínimo la distancia entre los puntos de ambas nubes. Este es un proceso largo y tedioso, con lo cual hay que poner un límite de iteraciones para que este proceso finalice en un tiempo limitado.

Para realizar este paso se utilizan las técnicas implementadas en la librería PCL.

Las técnicas probadas son las siguientes:

- ***Iterative Closest Point* (ICP)**: Técnica rápida para nubes de puntos que se encuentran en poses muy similares. Para poder aplicar esta técnica es necesario realizar un posicionamiento previo, que junte las nubes lo máximo posible. Para ello, se traslada la nube de entrenamiento hasta donde esté la nube de la escena, sumándole a los puntos de la nube de entrenamiento la diferencia de los centroides de ambas nubes.

  Esta técnica se explicó en el apartado 3.3.2.

- ***Random Sample Consensus Model* (RANSAC-M)**: Técnica más lenta, pero no influye la posición en la que se encuentren ambas nubes, haciendo que no sea necesario realizar un posicionamiento previo, pero esto no implica un menor tiempo de ejecución, todo lo contrario, el tiempo de ejecución de esta técnica es 14 veces más lenta que ICP, algo a tener en cuenta en la decisión final.

  Esta técnica se explicó en el apartado 3.3.1.

Ambas técnicas proporcionan una matriz de transformación desde el objeto de la escena hasta la nube del entrenamiento. Esta matriz es necesaria para poder realizar el siguiente paso. En el caso de utilizar el algoritmo ICP será necesario restarle a la pose de esta matriz el incremento realizado en el posicionamiento previo de la nube.

4.5.2. Cálculo de la pose del objeto

Para obtener la pose del objeto de la escena solo hay que realizar una simple multiplicación de matrices de transformación.

Las matrices de transformación necesarias son:

- La matriz de la base del robot a la vista guardada, almacenada en el XML de cada objeto (*base_a_vista*), representada en la imagen 4.5 con la flecha roja.
- La matriz de la vista guardada al objeto de la escena, obtenida en el proceso de ajuste de nubes (*vista_a_objecto*), representada en la imagen 4.5 con la flecha verde.

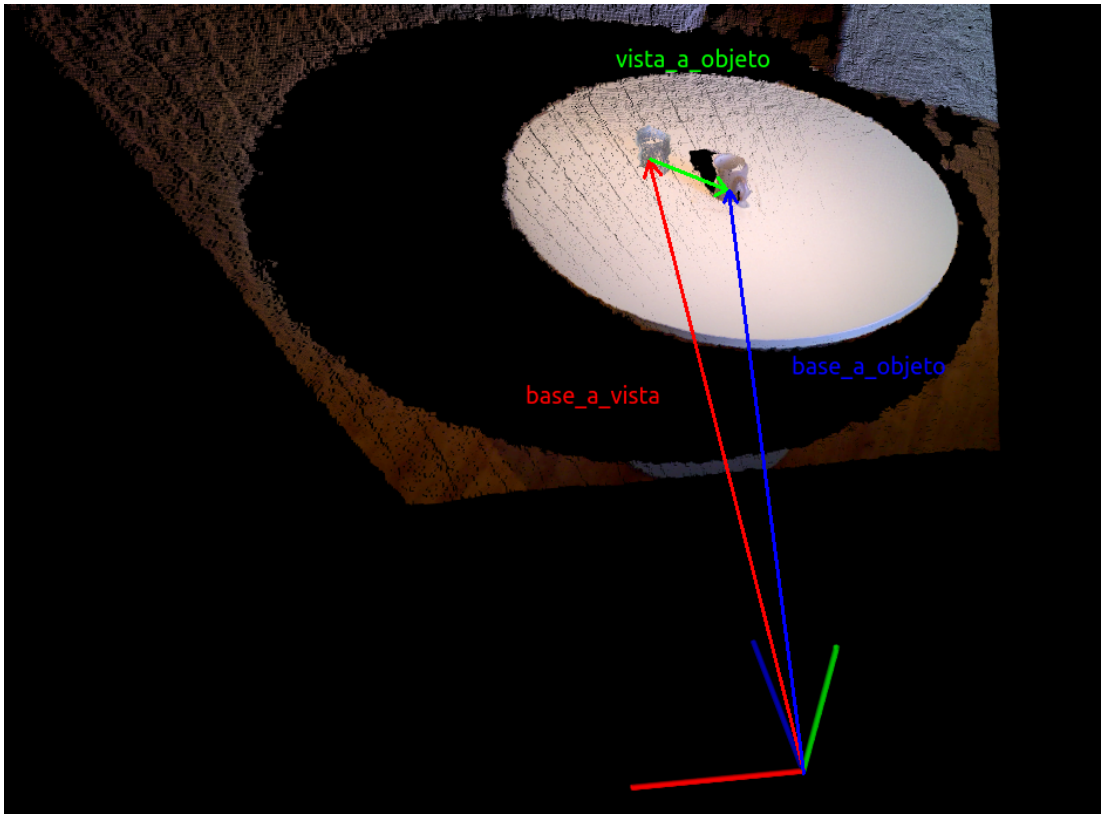

Figura 4.5: Representación del cálculo de la pose de un objeto con matrices de transformación

$$vista\_a\_objeto \times base\_a\_vista = base\_a\_objecto \tag{4.1}$$

La matriz obtenida en la ecuación 4.1 (*base_a_objecto*), es la matriz de transformación que lleva el un punto del sistema de referencia del robot al sistema de referencia del objeto, representada en la imagen 4.5 con la flecha azul.

De esta matriz resultante se extrae las coordenadas homogéneas, coincidiendo estas con la pose del objeto respecto al sistema de referencia de la base del robot.

4.5.3. Obtención del Cubo delimitador o *Bounding-box*

El cubo delimitador alineado con los ejes (o AABB), también llamado *Bounding-box*, es el cubo de menor tamaño que contiene los puntos de una nube de puntos alineado con los ejes *"x"*, *"y"* y *"z"*. El AABB es fácil de obtener gracias a la librería PCL en la que existe un método que calcula el AABB de una nube de puntos.

4.6. Optimización

Para que los resultados se obtengan con la mayor rapidez posible se aplicaron los siguientes cambios:

- **Reducción de la densidad de la escena.** En la entrada de la nube de puntos, con el fin de acelerar los procesos de tratamiento de la nube de puntos, se le aplica a la nube un filtro Voxel de 3mm. Si se aplica un filtro de tamaño superior se pierde demasiada información de los objetos. En la figura 4.6a, se puede observar la nube de puntos de la escena real, formada por 307200 puntos y en la figura 4.6b se puede ver la nube de puntos de la escena pasada por el filtro de red Voxel de 3mm, formada por 204372 puntos reduciéndose esta en un 33.47 %. Como se observa en esta imagen se ha disminuido la densidad de la nube sin llegar a perder información importante.

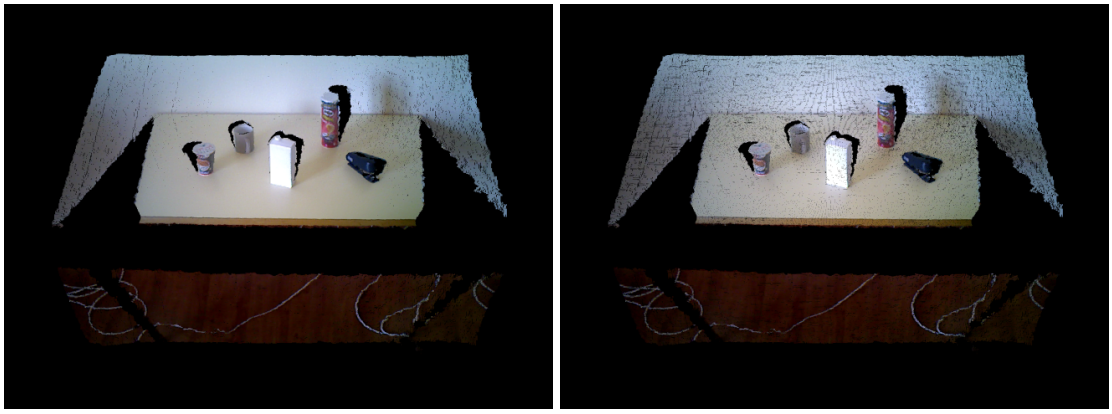

Figura 4.6: 4.6a Nube de puntos de la escena sin filtro Voxel. 4.6b Nube de puntos de la escena con filtro Voxel.

- **Eliminación de la parte inferior de la escena.** Partiendo de la suposición de que todos los objetos van ha estar colocados encima de una superficie plana, a una altura superior de 70cm, también con el fin de acelerar los procesos de tratamientos de la nube de puntos, se aplica a la nube de puntos un filtro que elimine todos los puntos que tengan un valor *"y"* inferior a este umbral. En la figura 4.7a, se puede ver la nube de puntos original de la escena, formada por 307200 puntos y en la figura 4.7b se observa que si se eliminan los puntos que se tienen un valor *"y"* inferior al umbral, se obtiene una nube formada por 180506 puntos, reduciéndose un 41.2 % el tamaño número de puntos de la escena, sin perder información significativa. Si se combinan ambas medidas, filtro Voxel y eliminación de la parte inferior, se obtiene una nube formada por 126545 puntos, reduciéndose la nube original en un 58.8 %.

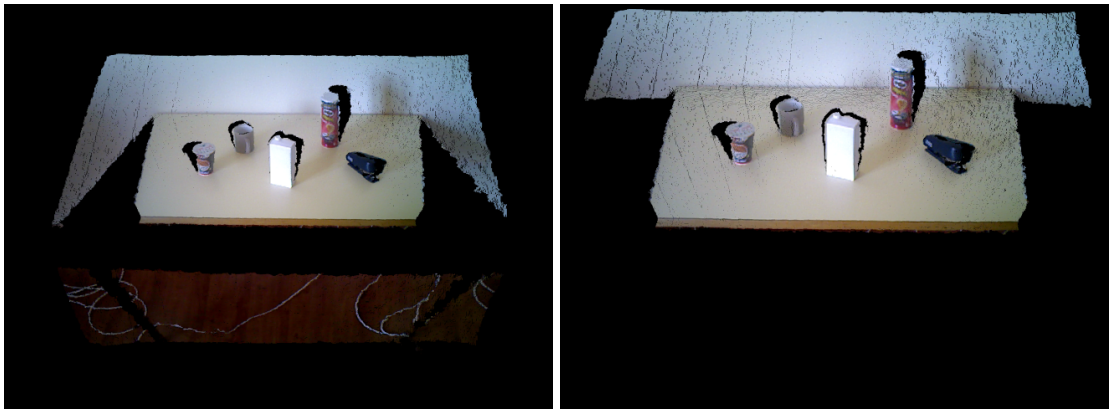

Figura 4.7: 4.7a Nube de puntos de la escena con suelo. 4.7b Nube de puntos de la escena sin suelo.

- **Uso del paralelismo con *threads*.** Una vez ya se dispone de los objetos segmentados, se realiza para todos y cada uno de ellos las mismas operaciones, *matching*, obtención del *Bounding-box*, *fitting* y obtención de la pose, este proceso se puede paralelizar, creando un hilo para cada objeto, reduciendo el tiempo de ejecución de manera muy significativa. Sin esta optimización el tiempo de ejecución medio de una escena con 5 objetos es de 14.3 s, sin embargo, si se aplican *threads* el tiempo medio de ejecución disminuye a 4.5 s, una variación considerable. Sin embargo, si se tiene una escena con 1 solo objeto el tiempo de ejecución sin la optimización es de 8.7 s y con la optimización se incrementaría a 8.9 s, debido al tiempo invertido en la creación de los hilos.

- **Uso del paralelismo con OpenMP.** Con el fin de obtener un menor tiempo de ejecución, se intenta acelerar el proceso de cálculo de bucles y otros procesos que podrían ser paralelos dentro de los threads anteriores, por ejemplo, el cálculo del *Bounding-box* y el cálculo de la pose. Finalmente y tras tomar medidas de los tiempos de ejecución, se decide aplicar esta medida solo al paralelismo de las secciones *Bounding-box* y cálculo de la pose, dado que el componente esta corriendo en un i7-5557U a 3.10GHz y en el que además de él corren diferentes componentes y se sobrecarga con demasiados hilos si se aplican ambas medidas. Si se tiene una escena con 1 solo objeto el tiempo de ejecución

sin la optimización es de 8.7 s y con la optimización se reduciría a 2.5 s.

Para que los resultados obtenidos sean lo más exactos o correctos posible, se decidió aplicar las siguientes medidas:

- **Ampliación de número de vistas del entrenamiento.** Cuantas más vistas existan de cada objeto, no solo se obtendrán mayor similitud en los descriptores a la hora de realizar el *matching*, sino que también se acelerará el proceso de *fitting* ya que las nubes estarán menos giradas. Los resultados de la ampliación del número de vistas se comentan en la sección 5.

- **Tomar vistas a diferentes distancias del robot.** Si un objeto se encuentra cerca del robot, este verá más la parte superior del objeto, mientras que si se encuentra lejos se visualizará mayormente la parte frontal. Un sistema que haya sido entrenado siempre a la misma distancia dará peores resultados en distancias lejanas a las que se ha entrenado. En la figura 4.8, podemos observar un ejemplo de entrenamiento.

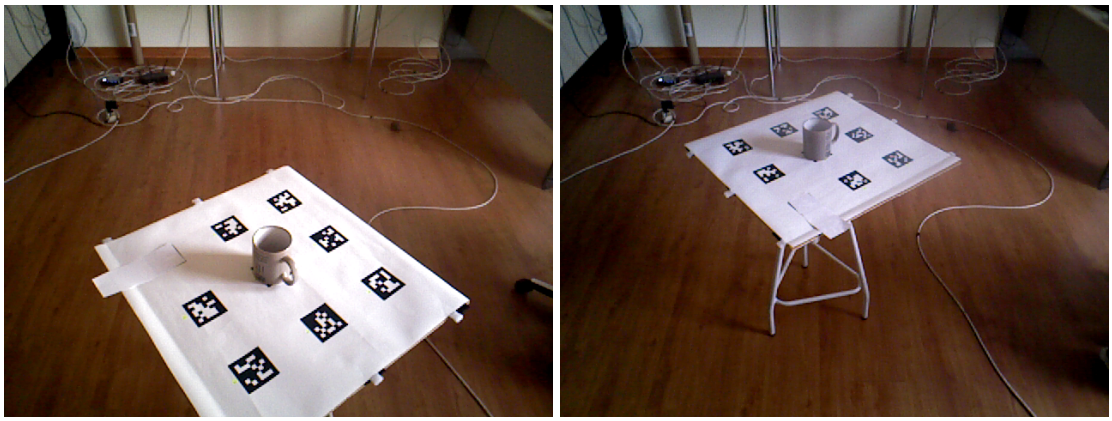

Figura 4.8: Ejemplo capturas del entrenamiento (Taza) 4.8a Taza cerca. 4.8b Taza lejos.

4.7. Integración del reconocimiento, el ajuste y la obtención de la pose

Una vez esté el sistema entrenado con diferentes objetos, si se invoca el método de su interfaz **findObjects**, este devolverá una lista de objetos, en la que cada objeto contendrá la etiqueta que lo clasifica **(Si no ha conseguido clasificarlo llevará la etiqueta *"unknown"*)**, la pose **(Si no se ha conseguido clasificar estará rellena con *0*)** y el *Bounding-box* que lo contiene.

De esta manera, se conocerán todos los objetos que hay en la escena, y en caso de que uno sea desconocido, se tendrá el *Bounding-box* que lo contiene, teniendo así una aproximación del tamaño de dicho objeto.

En la imagen 4.9, podemos visualizar la solución proporcionada por el sistema en una escena de prueba. Como se observa en esta figura se identifica la pared como un objeto, pero al no poder clasificarse le asigna la etiqueta *"unknown"*. Sin embargo, en el caso del brick, el *matcher* devuelve valores muy similares entre los descriptores de entrenamiento de pringles y brick, con lo cual le asigna la etiqueta *"unknown"* con el fin de no cometer un error.

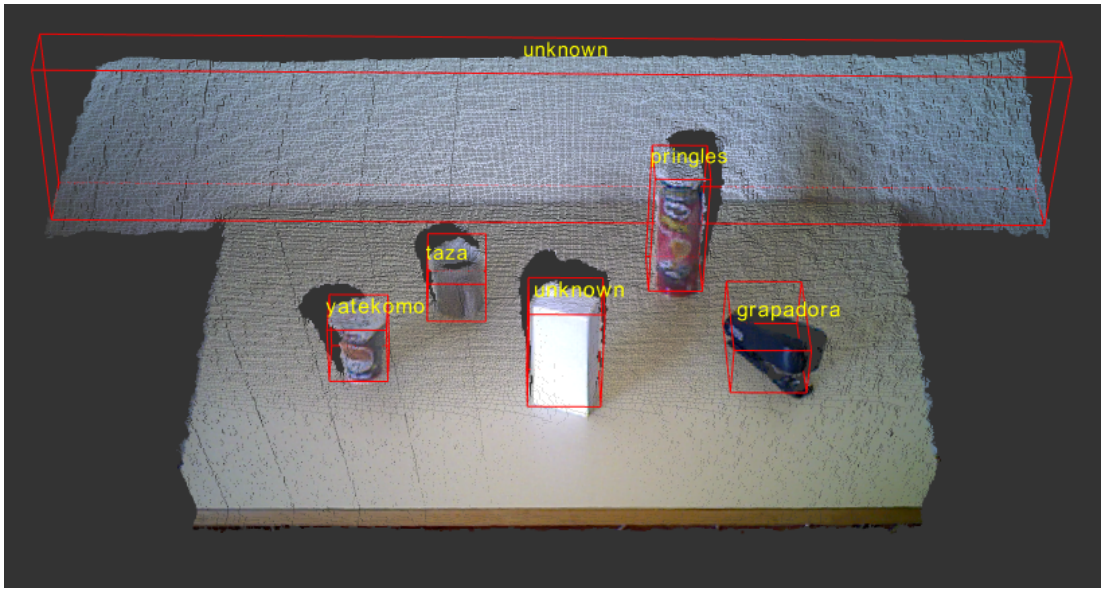

Figura 4.9: Solución Final Sistema de Reconocimiento

Capítulo 5

Experimentos

En este capítulo se expondrán los diferentes experimentos que se han realizado para probar la fiabilidad del sistema de reconocimiento y obtención de la pose de objetos.

5.1. Pruebas de reconocimiento

Un sistema de reconocimiento de objetos debe tener una alta tasa de aciertos, de esta manera, con esta prueba se pretende calcular el nivel de fiabilidad del sistema implementado.

Este experimento tiene dos variables independientes:

- Número de vistas usado en el entrenamiento: 6 (3 cerca y 3 lejos), 8 (4 cerca y 4 lejos), 12 (6 cerca y 6 lejos) y 16 (8 cerca y 8 lejos).
- Tipo de descriptor a usar (VFH, CVFH y OUR-CVFH).

Las variables de control son las siguientes:

- 50 nubes por objeto.
- Umbral de confusión del *matching* de 0.8. Este valor influirá significativamente en los resultados obtenidos.

Con estas pruebas se podrá concluir cual es la configuración óptima para sistema implementado.

Para cada configuración posible se ha obtenido la matriz de confusión del sistema de reconocimiento de objetos. De esta matriz se puede obtener mucha información, pero se centrará la atención en la tasa de acierto total (TA), la tasa de falso positivo total (TFP) y la tasa de desconocidos total (TD).

5.1.1. Resultados obtenidos con VFH

Para las tablas (5.1, 5.2, 5.3 y 5.4) se ha utilizado el descriptor VFH, y el número de vistas es 6, 8, 12 y 16 respectivamente.

En la tabla 5.1 se puede observar que el brick de leche tiene una alta tasa de falso positivo, esto se debe a la similitud de ambas nubes y al tipo de descriptor que se esta usando, ya que este tipo de descriptor tiene algunas deficiencias explicadas en el apartado 3.2.3. Por lo general, esta configuración tiene una alta tasa de aciertos, tiene un valor bajo de falsos positivos, y una tasa de desconocidos baja, pero estos valores pueden mejorarse con otras configuraciones.

Tabla 5.1: Matriz de confusión 6 vistas VFH

		Predicción					
		Brick	Grapadora	Pringles	Taza	Yatekomo	Desconocido
Real	Brick	21	0	19	0	0	10
	Grapadora	0	33	0	0	1	16
	Pringles	0	0	50	0	0	0
	Taza	0	0	0	45	0	5
	Yatekomo	0	0	0	0	50	0

TFP	TA	TD
8 %	79,6 %	12,4 %

En la tabla 5.2 se puede observar que la tasa de aciertos y la tasa de falsos positivos

empeoran, estos valores se deben a la forma del entrenamiento, es decir, las vistas que se han utilizado para entrenar el sistema. También es debido a las deficiencias que acarrea este tipo de descriptor.

Tabla 5.2: Matriz de confusión 8 vistas VFH

		Predicción					
		Brick	Grapadora	Pringles	Taza	Yatekomo	Desconocido
Real	Brick	19	0	20	0	0	11
	Grapadora	0	29	1	0	3	17
	Pringles	0	0	49	0	0	1
	Taza	0	0	0	48	0	2
	Yatekomo	0	0	0	0	50	0

TFP	TA	TD
9,6 %	78 %	12,4 %

En la tabla 5.3 se puede visualizar que al igual que en la configuración anterior la tasa de aciertos empeora, pero la tasa de falsos positivos disminuye, teniendo en cuenta que se prefiere decir que no se sabe que es un objeto a cometer un error, esta es una configuración mejor que las anteriores, pero aún existen configuraciones que mejoran estos resultados.

Tabla 5.3: Matriz de confusión 12 vistas VFH

		Predicción					
		Brick	Grapadora	Pringles	Taza	Yatekomo	Desconocido
Real	Brick	22	0	9	0	0	19
	Grapadora	0	31	0	0	3	16
	Pringles	0	0	42	0	0	8
	Taza	0	0	0	48	0	2
	Yatekomo	0	0	0	0	50	0

TFP	TA	TD
4,8 %	77,2 %	18 %

En la tabla 5.4 se observa que la tasa de aciertos ha aumentado significativamente, manteniéndose la tasa de falsos positivos como en la configuración utilizada en la tabla 5.3, aumentando la tasa de desconocidos, algo preferible.

Tabla 5.4: Matriz de confusión 16 vistas VFH

		Predicción					
		Brick	Grapadora	Pringles	Taza	Yatekomo	Desconocido
Real	Brick	22	0	10	0	0	18
	Grapadora	1	37	0	0	1	11
	Pringles	0	0	46	0	0	4
	Taza	0	0	0	48	0	2
	Yatekomo	0	0	0	0	50	0

TFP	TA	TD
4,8 %	81,2 %	14 %

Llegado el fin de este subexperimento, la configuración más adecuada para el sistema de reconocimiento de objetos, es el uso de 16 vistas para el entrenamiento.

5.1.2. Resultados obtenidos con CVFH

Para las tablas (5.5, 5.6, 5.7 y 5.8) se ha utilizado el descriptor CVFH, y el número de vistas es 6, 8, 12 y 16 respectivamente.

En la tabla 5.5 en comparación con la tabla 5.1 se puede observar que la tasa aciertos y falsos positivos disminuye, incrementando la tasa de desconocidos, así se puede afirmar que en igualdad de vistas, nuestro sistema es mejorado gracias a que CVFH solventa las deficiencias de VFH.

Tabla 5.5: Matriz de confusión 6 vistas CVFH

		Predicción					
		Brick	Grapadora	Pringles	Taza	Yatekomo	Desconocido
Real	Brick	20	2	10	0	1	18
	Grapadora	0	50	0	0	0	0
	Pringles	0	0	44	0	0	6
	Taza	0	0	0	43	0	7
	Yatekomo	0	0	0	0	38	12

TFP	TA	TD
5,2 %	78 %	17,2 %

En la tabla 5.6, se puede observar como aumentando el número de vistas respecto a la tabla 5.5 aumenta la tasa de aciertos y la tasa de falsos positivos esto es debido a la forma de entrenamiento que se ha utilizado, al igual que sucedía en la configuración de la tabla 5.2.

Tabla 5.6: Matriz de confusión 8 vistas CVFH

		Predicción					
		Brick	Grapadora	Pringles	Taza	Yatekomo	Desconocido
Real	Brick	17	0	18	0	0	15
	Grapadora	0	50	0	0	0	0
	Pringles	0	0	49	0	0	1
	Taza	0	0	0	45	0	5
	Yatekomo	0	0	0	0	45	5

TFP	TA	TD
7,2 %	82,4 %	10,4 %

En la tabla 5.7 se puede observar como respecto a la tabla 5.6 aumentando el número de vistas se ha reducido significativamente la tasa de falsos positivos, aumentando así mismo la tasa de aciertos y la tasa de desconocidos.

Tabla 5.7: Matriz de confusión 12 vistas CVFH

		Predicción					
		Brick	Grapadora	Pringles	Taza	Yatekomo	Desconocido
Real	Brick	23	0	5	0	1	21
	Grapadora	0	50	0	0	0	0
	Pringles	0	0	49	0	0	1
	Taza	0	0	0	42	0	8
	Yatekomo	0	0	0	0	47	3

TFP	TA	TD
2,4 %	84,4 %	13,2 %

En la tabla 5.8 respecto a la tabla 5.7 se puede visualizar como empeoran los resultados del sistema, de manera insignificativa, estos resultados se deben al valor

el umbral, pudiendo solventar este problema incrementando este valor.

Tabla 5.8: Matriz de confusión 16 vistas CVFH

		Predicción					
		Brick	Grapadora	Pringles	Taza	Yatekomo	Desconocido
Real	Brick	25	0	12	0	0	13
	Grapadora	0	50	0	0	0	0
	Pringles	0	0	40	0	0	10
	Taza	0	0	0	46	0	4
	Yatekomo	0	0	0	0	47	3

TFP	TA	TD
4,8 %	83,2 %	12 %

Una vez finalizado este segundo subexperimento, se puede decir que la mejor configuración interna a este subexperimento sería entrenando el sistema con 12 vistas, donde obtenemos la menor tasa de falsos positivos y la mayor tasa de aciertos.

5.1.3. Resultados obtenidos con OUR-CVFH

Para las tablas (5.9, 5.10, 5.11 y 5.12) se ha utilizado el descriptor OUR-CVFH, y el número de vistas es 6, 8, 12 y 16 respectivamente.

En la tabla 5.9 se puede observar que con este tipo de descriptor la tasa de aciertos es muy baja respecto a las pruebas realizadas con VFH y CVFH, este valor tan bajo es debido al umbral configurado para etiquetar un objeto como desconocido. De esta manera, elevando este umbral, conseguiríamos incrementar la tasa de aciertos y reducir así la tasa de desconocidos ocasionando también un incremento en la tasa de falsos positivos.

Tabla 5.9: Matriz de confusión 6 vistas OUR-CVFH

		Predicción					
		Brick	Grapadora	Pringles	Taza	Yatekomo	Desconocido
Real	Brick	15	0	5	0	0	30
	Grapadora	0	34	0	0	0	16
	Pringles	0	0	38	0	0	12
	Taza	0	0	0	30	1	19
	Yatekomo	0	10	0	0	8	28

TFP	TA	TD
8 %	50 %	42 %

En la tabla 5.10 se puede observar como al igual que en la tabla 5.9 el valor de la tasa de aciertos es muy bajo, pero al incrementar el número de vistas aumenta la tasa de aciertos de manera significativa, mejorando los resultados de la tabla 5.9.

Tabla 5.10: Matriz de confusión 8 vistas OUR-CVFH

		Predicción					
		Brick	Grapadora	Pringles	Taza	Yatekomo	Desconocido
Real	Brick	22	0	12	1	0	15
	Grapadora	0	39	0	0	0	11
	Pringles	0	0	36	0	0	14
	Taza	0	0	0	32	0	18
	Yatekomo	0	0	0	0	24	26

TFP	TA	TD
5,2 %	61,2 %	33,6 %

En la tabla 5.11 al igual que en la tabla 5.10 se visualiza como aumenta la tasa de aciertos, con el aumento del número de vistas, obteniendo así una tasa de falsos

positivos muy reducida, aumentando de manera insignificativa la tasa de desconocidos.

Tabla 5.11: Matriz de confusión 12 vistas OUR-CVFH

		Predicción					
		Brick	Grapadora	Pringles	Taza	Yatekomo	Desconocido
Real	Brick	26	0	3	0	0	21
	Grapadora	0	43	0	0	0	7
	Pringles	0	0	36	0	0	14
	Taza	0	0	0	32	0	18
	Yatekomo	0	0	0	0	24	25

TFP	TA	TD
1,6 %	64,4 %	34

En la tabla 5.12 se obtiene la menor tasa de falsos positivos y se disminuye la tasa de desconocidos respecto a la tabla 5.11, obteniendo así una mayor tasa de aciertos.

Tabla 5.12: Matriz de confusión 16 vistas OUR-CVFH

		Predicción					
		Brick	Grapadora	Pringles	Taza	Yatekomo	Desconocido
Real	Brick	22	0	1	0	0	27
	Grapadora	0	49	0	0	0	1
	Pringles	0	0	36	0	1	13
	Taza	0	0	0	31	0	19
	Yatekomo	0	0	0	0	33	17

TFP	TA	TD
0,8 %	68,4 %	30,8 %

Una vez finalizado este subexperimento, se puede decir que la mejor configuración interna a este subexperimento sería entrenando el sistema con 16 vistas, obteniéndose

así una tasa de falsos positivos inferior al 1 %.

5.1.4. Conclusiones respecto al reconocimiento

Respecto al experimento completo, se concluye que el valor del umbral influye significativamente en los resultados obtenidos. Dependiendo de los resultados que se desea obtener, se puede decir que:

- El descriptor más adecuado para obtener una menor tasa de falsos positivos sería OUR-CVFH.
- El descriptor más adecuado para obtener una mayor tasa de aciertos es CVFH.

En general incrementando el número de vistas se ha mejorado el sistema. Además, el número de vistas no solo es importante para el proceso de *matching*, ya que se trata de un sistema de reconocimiento y obtención de la pose, donde el número de vistas influye también a la hora de realizar el *fitting*, como se explicó en el apartado 4.6.

5.2. Pruebas de detección de pose

Dado que el sistema final no solo está dedicado a clasificar objetos, sino que también calcula la pose del objeto, es necesario realizar una batería de pruebas teniéndose como objeto de estudio el cálculo de la pose. Para estas pruebas se obtendrá la posición de diferentes objetos y se medirá el error cometido en cada una de las iteraciones.

Como existen dos tipos de algoritmos para realizar el *fitting*, se realizarán las mismas pruebas para ambos algoritmos. Los datos del experimento son:

- *Matching* con VFH.
- 50 escenas por objeto
- Forzar los resultados del *matching* entrenando el sistema con un solo objeto (ej. en la escena hay una taza, el *train* solo contiene vistas de la taza).

- Comparación de la correctitud del resultado de forma visual (Subjetivo).
- Pruebas de la técnica ICP y de RANSAC, con diferentes entrenamientos.

5.2.1. Resultados obtenidos con RANSAC

Como se puede observar en la tabla 5.13, la técnica RANSAC a pesar de no necesitar un posicionamiento previo de las nubes de puntos, se trata de un algoritmo que tiene una menor probabilidad de dar la pose correcta. Como también se ha probado este algoritmo con diferentes números de vistas en el entrenamiento, se puede observar como si se incrementa el número de vistas la probabilidad de que el sistema calcule la pose correcta incrementa significativamente. Además, como ya se comentó, su tiempo de cómputo es bastante alto.

Tabla 5.13: Pruebas ajuste de nube de puntos

		ICP		RANSAC	
Nº Vistas	**Objeto**	**Aciertos**	**Errores**	**Aciertos**	**Errores**
16	Yatekomo	50	0	41	9
	Taza	46	4	34	16
	Pringles	50	0	41	9
	Grapadora	40	10	30	20
	Brick	50	0	43	7
	Total	**236**	**14**	**189**	**61**
12	Yatekomo	50	0	40	10
	Taza	44	6	32	18
	Pringles	50	0	39	11
	Grapadora	38	12	25	25
	Brick	48	2	41	9
	Total	**230**	**20**	**177**	**73**
8	Yatekomo	50	0	42	8
	Taza	43	7	32	18
	Pringles	49	1	38	12
	Grapadora	36	14	21	29
	Brick	45	5	39	11
	Total	**223**	**27**	**172**	**78**
6	Yatekomo	50	0	40	10
	Taza	40	10	30	20
	Pringles	49	1	39	11
	Grapadora	34	16	14	36
	Brick	41	9	34	16
	Total	**214**	**36**	**157**	**93**

5.2.2. Resultados obtenidos con ICP

Como se puede observar en la tabla 5.13, la técnica ICP a diferencia de RANSAC aunque necesita un posicionamiento previo de las nubes de puntos, tiene una mayor probabilidad de calcular la pose correcta. Al igual que con RANSAC se ha probado este algoritmo con diferentes números de vistas en el entrenamiento y se puede observar como si se incrementa el número de vistas la probabilidad de que el sistema calcule la pose correcta aumenta significativamente. Además, otra diferencia con RANSAC es que su tiempo de cómputo es mucho menor, estos no han sido medidos.

5.2.3. Conclusiones respecto al ajuste de la pose

De esta batería de pruebas se puede concluir que la técnica de ajuste de nubes de puntos más adecuada para un sistema de obtención de la pose sería ICP, teniendo este una probabilidad del 94,4 % frente a la probabilidad de RANSAC que es del 75,6 %. Además, como ya se comentó en secciones anteriores, tener un mayor número de vistas de los objetos, hace que estos algoritmos incrementen su probabilidad de acierto. En estas pruebas se ha obtenido con 6 vistas una probabilidad del 85.6 % y añadiendo 10 vistas más de los objetos se incrementa hasta el 94,4 % una diferencia significativa.

Capítulo 6

Conclusiones y trabajos futuros

Como se mencionó en la introducción los seres humanos cada vez confían más tareas en los robots y en este proyecto se ha explicado una posible implementación de un sistema con el que se resuelven las preguntas ***¿Cómo puede un robot reconocer un objeto?*** y ***¿Cómo puede un robot localizar el objeto?***

Como se puede apreciar en las pruebas, escoger un descriptor adecuado implicará tener unos mejores resultados de reconocimiento de objetos y escoger la técnica de *fitting* adecuada, también implica calcular la pose correcta del objeto un mayor número de veces, además de suponer una variación significativa en el tiempo computacional.

También se puede decir que para reducir el tiempo de cómputo lo máximo posible hay que conocer el entorno en el que se va ha ejecutar el programa, ya que no solo importa el número de cores del que dispone la máquina, sino también de la carga computacional que esta tiene.

Otra conclusión que hay que mencionar es que toda la información de la que disponga el robot de la escena es importante, ya que si este quiere coger un objeto y el sistema solo le aporta la información de este objeto y no del entorno que le rodea, podría existir otro objeto que impida que el robot pueda coger el objeto deseado. Por ello, es importante devolver la pose, la etiqueta y/o el *Bounding-box* de todos los objetos de la escena.

A modo de resumen, se puede decir que el sistema que se ha implementado obtiene unos resultados con una alta fiabilidad, como se ha podido comprobar en las pruebas que se han realizado con las diferentes configuraciones.

Para el sistema que se ha implementado en el robot Shelly se proponen los siguientes trabajos futuros:

- **Ejecución continua del detector de objetos:** Actualmente el sistema que se ha implementado no se ejecuta lo suficientemente rápido como para ejecutar el reconocimiento de objetos de forma continua. Este supondrá eliminar así el error producido en la odometría al patinar las ruedas sobre una superficie pulida o en caso de que el objeto se mueva de posición saber que ya no se encuentra ahí.

- **Seguimiento de los objetos (*Tracking*):** Realizar un seguimiento de los objetos de forma que se actualice la pose en caso de que este o el robot se muevan.

- **Imaginarse la pose de un objeto:** Con implementación del *Tracking* podría imaginarse la pose de un objeto, conociendo los movimientos que ha realizado Shelly.

Estos trabajos futuros supondrán que Shelly tenga una mayor tasa de aciertos a la hora de coger un objeto, ya que el sistema obtendrá continuamente la pose de los objetos.

Bibliografía

[1] Pedro Núnez, Luis J Manso, Pablo Bustos, Paulo Drews-Jr, and Douglas G Macharet. Towards a new semantic social navigation paradigm for autonomous robots using cortex.

[2] Boris Mederos, Luiz Velho, and Luiz Henrique De Figueiredo. Moving least squares multiresolution surface approximation. In *Computer Graphics and Image Processing, 2003. SIBGRAPI 2003. XVI Brazilian Symposium on*, pages 19–26. IEEE, 2003.

[3] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[4] Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. *Numerische mathematik*, 14(5):403–420, 1970.

[5] Carlos Mateo Agulló, Pablo Gil, and Fernando Torres. Detección de deformaciones 3d calculando esqueletos de curvaturas. 2015.

[6] Luís A Alexandre. 3d descriptors for object and category recognition: a comparative evaluation. In *Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vilamoura, Portugal*, volume 1, page 7, 2012.

[7] Radu Bogdan Rusu, Gary Bradski, Romain Thibaux, and John Hsu. Fast 3d recognition and pose using the viewpoint feature histogram. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2155–2162. IEEE, 2010.

[8] Gonzalo; Escalera Arturo de la (eds.) Alegre, Enrique; Pajares. *Conceptos y Métodos en Visión por Computador*. Wiley Online Library, 2016.

[9] Aitor Aldoma, Federico Tombari, Radu Rusu, and Markus Vincze. Our-cvfh–oriented, unique and repeatable clustered viewpoint feature histogram for object recognition and 6dof pose estimation. *Pattern Recognition*, pages 113–122, 2012.

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

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

[12] Dmitry Chetverikov, Dmitry Svirko, Dmitry Stepanov, and Pavel Krsek. The trimmed iterative closest point algorithm. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 3, pages 545–548. IEEE, 2002.

[13] Euclidean Cluster Extraction. `www.pointclouds.org/documentation/tutorials/cluster_extraction.php`.

[14] Luis Manso, Pilar Bachiller, Pablo Bustos, Pedro Núnez, Ramón Cintas, and Luis Calderita. Robocomp: a tool-based robotics framework. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2010.

[15] Marco Antonio Gutierrez Giraldo, Adrián Romero Garcés, Pablo Bustos García de Castro, and Jesús Martínez Cruz. Progress in robocomp. 2013.

[16] P Bustos, LJ Manso, MA Gutiérrez, M Haut, M Paoletti, I Garcıa-Varea, J Martınez-Gómez, L Rodrıguez-Ruız, A Romero-Garcés, and R Marfil. Ursus team.