Instituto Tecnológico de Costa Rica

Área Académica de Ingeniería Mecatrónica

Implementación de un sistema para la asistencia de pacientes mediante el robot Pepper.

Informe de Proyecto de Graduación para optar por el título de

Ingeniero en Mecatrónica con el grado académico de Licenciatura

Steven Antonio Jiménez Bustamante.

Cartago, 27 de abril de 2018

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Steven Antonio Jiménez Bustamante.

Cartago, 27 de abril de 2018

Céd: 2-0729-0765

Instituto Tecnológico de Costa Rica
Área Académica de Ingeniería Mecatrónica
Proyecto de Graduación
Tribunal Evaluador

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Mecatrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal

Dra. Ing. Gabriela Ortiz León
Profesora Lectora

MSc. Ing. Sergio Arriola Valverde
Profesor Lector

MSc. Ing. Yeiner Arias Esquivel
Profesor Asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por el Área Académica de Ingeniería Mecatrónica.

Cartago, 27 de abril de 2018

Instituto Tecnológico de Costa Rica
Área Académica de Ingeniería Mecatrónica
Proyecto de Graduación
Tribunal Evaluador
Acta de Evaluación

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Mecatrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Estudiante: Steven Antonio Jiménez Bustamante.

Nombre del Proyecto: *Implementación de un sistema para la asistencia de pacientes mediante el robot Pepper.*

Miembros del Tribunal

Dra. Ing. Gabriela Ortiz León Profesora Lectora	MSc. Ing. Sergio Arriola Valverde Profesor Lector

MSc. Ing. Yeiner Arias Esquivel
Profesor Asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por el Área Académica de Ingeniería Mecatrónica.

Nota final del Proyecto de Graduación: _____

Cartago, 27 de abril de 2018

Resumen

En los siguientes apartados se muestra el desarrollo de un ambicioso proyecto que consiste en utilizar el robot Pepper para que asista a pacientes discapacitados con el objetivo de mejorar su calidad de vida. Esto se realiza mediante la integración de herramientas de distintos campos de investigación como *deep learning*, reconocimiento de voz y manipulación de robots. Al final de este documento se demuestra que es posible incorporar este tipo de robots en beneficio de la sociedad y especialmente de los pacientes, aunque se deduce que hay varios detalles a nivel de programación y construcción del robot que deben ser mejorados.

Palabras clave: Faster R-CNN, Reconocimiento de Voz, ROS, Robot Pepper.

Abstract

The following sections show the development of an ambitious project that uses the Pepper robot to assist disabled patients with the purpose of improving their quality of life. To achieve this goal, it is necessary the integration of different tools from different fields of research such as deep learning, speech recognition and robot manipulation. At the end of this document it is shown that it is possible to incorporate this type of robots for the benefit of society and especially of patients, although with some restrictinos, because there are several details at the level of programming and construction of the robot that should be improved.

Keywords: Faster R-CNN, Speech Recognition, ROS, Robot Pepper.

a mi querida familia ...

Agradecimientos

Agradezco el apoyo incondicional que me han dado mis padres Jeannette y Uriel, y mi hermana Priscilla.

Gracias a mi tía Sulmar y François por todo el apoyo que me brindaron durante mi estadía en España.

Gracias a todos los que me ayudaron en el laboratorio RoViT principalmente a mi tutor Miguel pero también a Fran, Jose Carlos, Sergio, Zuri, Edmanuel, Nadia, Felix y Manuel.

Steven Antonio Jiménez Bustamante.

Cartago, 1 de mayo de 2018

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En el primer capítulo, se brinda una breve descripción del problema que se atacó con este proyecto, el contexto en el que se encuentra inmerso, una propuesta inicial de solución, los objetivos generales y específicos que definen los límites por alcanzar, la metodología de diseño utilizada y, por último, un párrafo que sintetiza cada uno de los capítulos que componen el documento.

1.1 Situación mundial

Es claro que en todos los países existen personas que, por razones de enfermedad, accidentes, vejez u otras situaciones, son declaradas con algún tipo de discapacidad y que, de manera parcial o total, pierden la movilidad en su cuerpo. A manera de ejemplo y basado en el resumen del Informe Mundial de la Discapacidad del año 2011 [1], se expresa lo siguiente:

> *Se estima que más de mil millones de personas viven con algún tipo de discapacidad, o sea, alrededor del 15% de la población mundial (según las estimaciones de la población mundial en 2010). Esta cifra es superior a las estimaciones previas de la Organización Mundial de la Salud, correspondientes a los años 1970, que eran de aproximadamente un 10%.*

Lo anterior permite afirmar que una gran cantidad de personas alrededor del mundo sufre de alguna discapacidad, para las cuales se pueden buscar alternativas, con el fin de garantizarles una mejor calidad de vida. Una de las prioridades de los profesionales en ingeniería es buscar el bienestar de los demás y lograr una vida digna para ellos, por esta razón, el problema que afrontó este proyecto fue la dificultad de ejecución de actividades cotidianas por parte de personas con algún tipo de discapacidad y, en este caso, se enfocó en la tarea de alcanzar objetos que se encuentran en su entorno. Afecciones como lesión cerebral y movilidad reducida congénita o adquirida son algunos de los casos clínicos que se apoyaron con este proyecto.

1.2 Implementación de un robot humanoide para afrontar el problema

Aunque el trabajo de atención de pacientes en situación de discapacidad puede ser ejecutado por personas capacitadas en el área, también puede ser realizado por robots capaces de cumplir con labores varias y, de esta forma, se podría reubicar al personal capacitado en actividades que requieran habilidades más complejas, con el fin de mejorar la atención a los pacientes. Como respuesta, se propuso implementar un sistema para la asistencia de pacientes mediante el robot Pepper, un robot humanoide capaz de asistir al paciente facilitando el cuidado que él requiere.

Debido a que el proyecto utilizó este robot, no hubo opciones de selección de *hardware*, sino que fue necesario aprovechar todo el potencial de dicho robot e incorporar todas las herramientas de *software* para ejecutar el proyecto de la mejor manera. Las especificaciones generales sobre el robot se muestran en la tabla A.1 y en la figura 1.1 se muestra una fotografía del robot Pepper, donde se pueden observar algunos de los módulos mencionados en dicha tabla.

El proyecto se llevó a cabo en el Laboratorio de Investigación en Robótica y Visión Tridimensional (RoViT), ubicado en la Universidad de Alicante, España. Este laboratorio investiga muchos aspectos de la robótica móvil, usando datos 3D como la principal fuente de percepción. Algunas de las líneas de investigación del laboratorio, relevantes a este proyecto, comprenden métodos tradicionales de visión por computador, aceleración de algoritmos utilizando GP-GPU y aprendizaje profundo. Por su parte, este proyecto supone una parte fundamental en los temas de investigación del laboratorio, ya que llega a reforzar el proyecto Retogar [2], el cual propone implementar un asistente robótico que realice tareas de asistencia a personas dependientes. El robot debe ser capaz de procesar la orden del paciente, reconocer y localizar el objeto solicitado, acercarse a él, agarrarlo y, por último, llevárselo a su sitio.

Utilizando todas estas herramientas que posee el robot, se propuso una solución que se describe seguidamente:

El robot debe estar pendiente de las instrucciones que le indique el paciente, para lo cual, se utilizarán los micrófonos, con el fin de grabar el audio (mensaje) emitido por la persona para, posteriormente, convertirlo a texto y, por último, interpretar este texto como una instrucción. Después de interpretar la instrucción, el robot debe buscar el objeto que se le solicitó. Si no logra localizarlo, este deberá notificarle al paciente que no lo encontró; por otro lado, si logra localizarlo, este deberá desplazarse hasta donde se encuentre el elemento, sujetarlo y llevarlo hasta el paciente. El robot debe entregarle el objeto a la persona en un lugar cercano, por ejemplo, una mesa de noche, por motivos de seguridad y salud del paciente.

En la figura 1.2 se brinda un diagrama que describe la propuesta de solución inicial.

Figura 1.1: Fotografía del robot Pepper. [3]

1.3 Objetivos de la investigación

1.3.1 Objetivo general:

Diseñar un sistema que asista al paciente en la tarea específica de alcanzar objetos, según lo requiera la persona, utilizando el robot Pepper.

1.3.2 Objetivos específicos:

- Implementar el reconocimiento de los objetos específicos en el entorno que rodea al robot.
  - Indicador: el robot debe reconocer tres objetos pequeños, mínimo en el 90 % de los intentos, en un rango de uno a cinco metros de distancia.
- Implementar el reconocimiento de la voz del paciente y traducirlo a texto como una instrucción.
  - Indicador: el robot debe reconocer las siguientes palabras: traer, alcanzar y los tres nombres de los objetos que se deben reconocer, mínimo en un 90 % de los intentos, cuando esté a un metro de distancia o menos del paciente, una vez que el mensaje haya sido traducido de audio a texto.
- Integrar las tareas de desplazamiento y el movimiento de objetos utilizando el robot, junto con el reconocimiento de objetos y de voz.
  - Entregable: esta integración debe cumplir con las métricas definidas anteriormente, además de solucionar el problema de la siguiente forma: cuando la

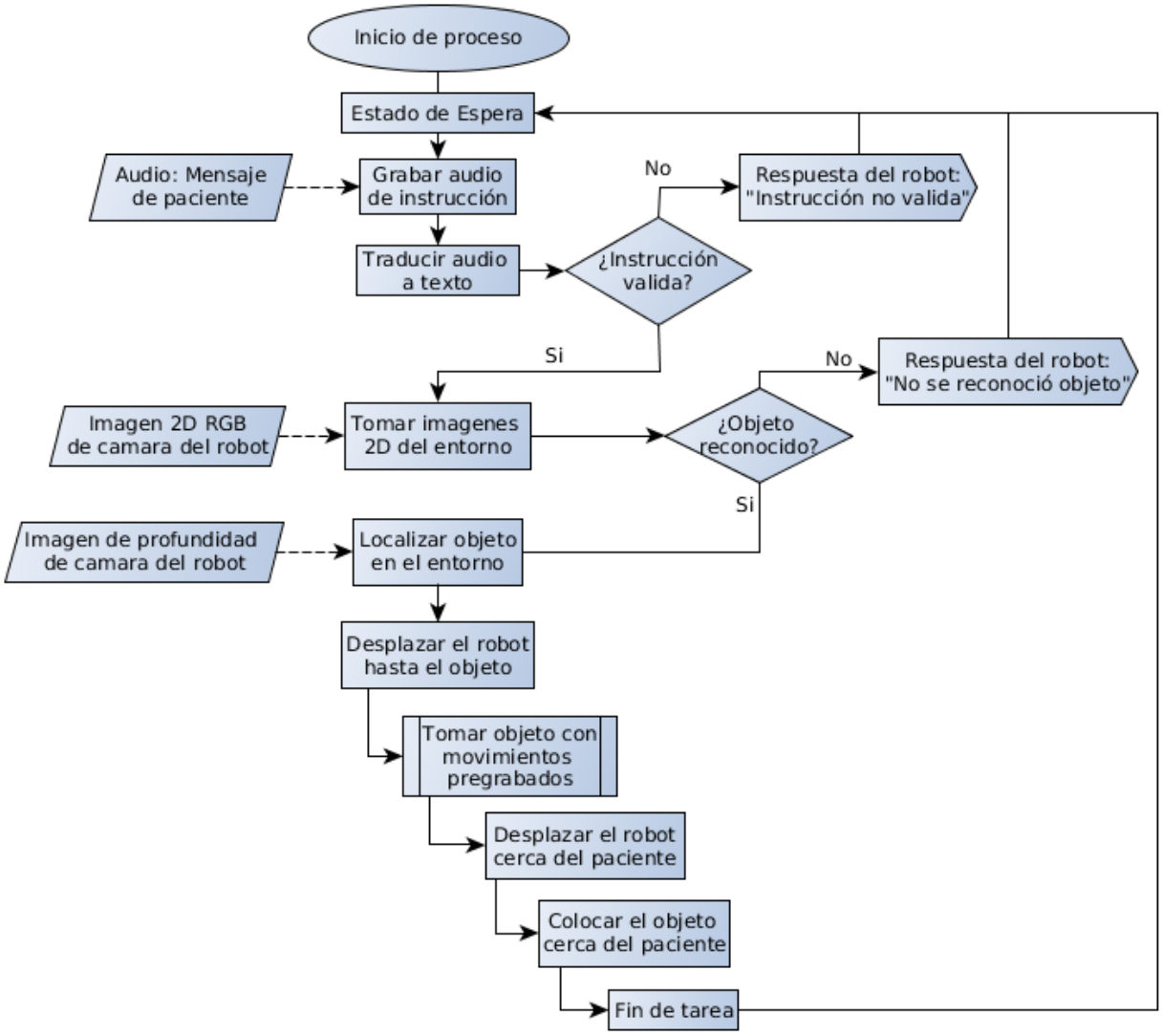


Figura 1.2: Diagrama de propuesta de solución.

instrucción haya sido interpretada, el robot debe buscar el objeto deseado, determinar su ubicación y desplazarse hasta este a una distancia de aproximadamente 35 centímetros, después, deberá tomarlo y llevarlo hasta una posición cerca del paciente, de igual manera a una distancia aproximada de 35 centímetros. Si el robot no logra ubicar el objeto, deberá indicarle al paciente que no logró identificarlo.

1.4 Metodología utilizada en la investigación

El éxito que tienen las ingenierías para solucionar problemas se basa en la creatividad y propuestas de soluciones con fundamentos científicos. Aunque hay muchas formas para solucionar problemas en distintos ámbitos, en el ámbito ingenieril, la metodología tiene como base una serie de pasos, que consisten en primero desarrollar una descripción clara y

concisa del problema; identificar, al menos de manera tentativa, los factores importantes que afectan el problema o que pueden jugar un papel en la solución; proponer un modelo para el problema, utilizando los conocimientos científicos o de la ingeniería del fenómeno bajo estudio, tomando en cuenta todas las limitaciones o supuestos del modelo.

Además, el siguiente paso consiste en realizar los experimentos apropiados y recolectar datos para probar o validar el modelo tentativo; refinar el modelo con base en los datos observados; manipular el modelo para contribuir a desarrollar una solución del problema; realizar un experimento para confirmar que la solución propuesta del problema es efectiva y al mismo tiempo eficiente; por último, obtener conclusiones o hacer recomendaciones con base en la solución del problema [4]. En el diagrama mostrado en la figura 1.3, se muestran los pasos que sigue este proyecto en busca de la solución.

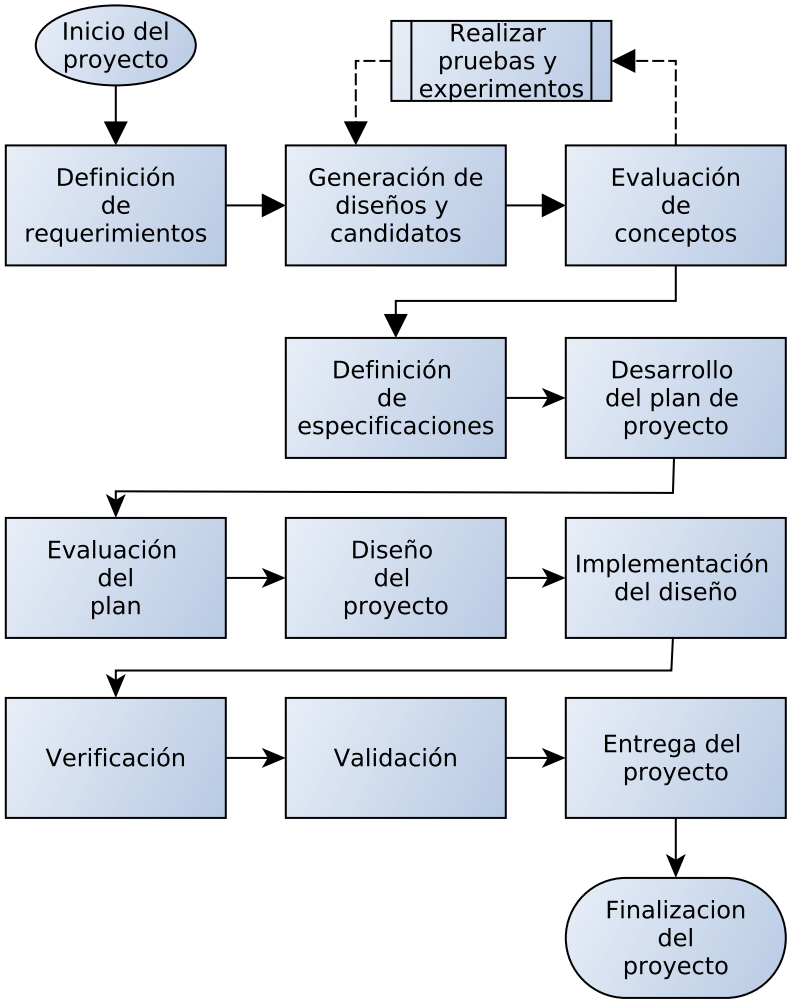


Figura 1.3: Metodología de investigación aplicada al proyecto.

A continuación, se muestra, en pocas palabras, en qué consisten las etapas de la metodología de la investigación utilizada en este proyecto:

1. Definir los requerimientos del proyecto: los requerimientos describen las necesidades

que debe satisfacer el proyecto.

2. Generar los candidatos y estructuras posibles del proyecto: según una serie de diseños propuestos, se evalúan los conceptos y se establece cuál o cuáles deben continuar y cuáles deben morir. Esto se comporta como un proceso iterativo, en busca de las mejores propuestas.

3. Definir las especificaciones del proyecto: las especificaciones establecen de forma clara todas las características que debe reunir un proyecto para que sea funcional. Esta etapa es realmente crítica, ya que permite ahorrar tiempo y trabajo, debido a que delimita los alcances del proyecto.

4. Desarrollar el plan de proyecto: este plan marca el ritmo de trabajo y metas por lograr, a corto, mediano y largo plazo.

5. Ejecutar el plan de proyecto: se pone en marcha el plan del proyecto para lograr los objetivos planteados.

6. Diseñar e implementar: en esta etapa, se define el diseño y, con base en este, se inicia con su implementación o desarrollo.

7. Verificar el proyecto: basado en una lista de verificación propuesta, idealmente, por un grupo evaluador del proyecto, si algún requerimiento no se cumple, se debe regresar a etapas previas y solventar el problema.

8. Validar el proyecto: basado en una lista de verificación propuesta, idealmente, por un grupo evaluador del proyecto, si alguna especificación no se cumple, se debe regresar a etapas previas y solventar el problema.

9. Entrega: en este apartado se entrega el proyecto, además, se brindan las conclusiones y recomendaciones que arroja el desarrollo del mismo.

1.5 Estructura del documento

Este documento consta de cinco capítulos, en el capítulo 2 se detalla el estado del arte en el que se fundamenta este trabajo. En el capítulo 3, se muestra la solución implementada y la forma como se lograron los objetivos propuestos y bajo cuáles condiciones. En el capítulo 4, se brindan los resultados obtenidos y su respectivo análisis. En el capítulo 5, se concluye sobre el trabajo realizado y se aconseja al lector cuáles recomendaciones implementar. Por último, se muestra la sección de referencias bibliográficas y los anexos.

Capítulo 2

Marco teórico

En este apartado, se exponen los fundamentos teóricos que fueron utilizados para solucionar el problema, los cuales giran alrededor de temas como visión por computador, aprendizaje profundo, cámaras 3D, reconocimiento de voz, así como el *software* utilizado para integrarlas. Además, se brinda información sobre el robot Pepper, que es la pieza fundamental en este proyecto, ya que permite que sea un prototipo implementado en la realidad, en vez de quedar a nivel de simulación.

2.1 Visión por computador

La visión por computador es un campo de estudio de la ciencia que incluye métodos para adquirir, procesar, analizar y comprender las imágenes del mundo físico, con el objetivo de extraer información útil, para que sea procesada por un computador. Es sorprendente cómo los animales y los seres humanos llevan a cabo esta tarea con poco esfuerzo, mientras que implementarla en un computador se convierte en un sistema altamente propenso a cometer errores [5].

La detección de objetos es un área dentro del campo de visión por computador, la cual estudia cómo detectar la presencia de objetos en una imagen basándose en su apariencia, por ejemplo, una persona o un automóvil. Generalmente, se pueden distinguir dos partes en el proceso de detección: la extracción de características del contenido de una imagen y la búsqueda de objetos basada en dichas características.

En la sección 2.2, se muestra la base sobre cómo se llevó a cabo el reconocimiento de objetos en este proyecto, utilizando aprendizaje profundo y dejando de lado las técnicas tradicionales de visión por computador.

2.2 Aprendizaje profundo

Para comprender lo que es aprendizaje profundo, es necesario describir los conceptos de inteligencia artificial y aprendizaje automático.

- Inteligencia artificial: se define como un comportamiento inteligente adoptado por las máquinas, a diferencia de la inteligencia natural de los seres humanos y animales. En las ciencias de computación, el término inteligencia artificial se refiere al estudio de "agentes inteligentes", los cuales se definen como cualquier dispositivo que logre percibir su entorno y tome decisiones que maximicen las posibilidades de cumplir exitosamente una tarea.
- Aprendizaje automático: es un campo de estudio de las ciencias de la computación, el cual brinda habilidades a las computadoras para aprender sin haber sido explícitamente programadas.

Teniendo claros esos términos, el aprendizaje profundo se encarga de explorar el estudio y construcción de algoritmos que pueden aprender desde un conjunto de datos y hacer predicciones sobre estos. Tales algoritmos superan las instrucciones estrictamente estáticas de los programas convencionales, haciendo predicciones o decisiones basadas en datos, mediante la construcción de modelos basados en datos de entrenamiento.

En los últimos años, la inteligencia artificial ha crecido exponencialmente, en especial desde el año 2015. Mucho de lo que se ha logrado hasta ahora se debe a la disponibilidad de las GPU (en inglés *Graphical Processing Unit*), las cuales poseen una arquitectura robusta para procesar tareas computacionales difíciles. Además, no se puede dejar de lado que la inteligencia artificial ha sido posible gracias al movimiento del *Big Data*, donde grandes cantidades de información como imágenes, textos, transacciones, datos de mapeo, entre otros, sirven como conjuntos de datos de entrada para la etapa de aprendizaje.

La base del aprendizaje automático fueron las redes neuronales, las cuales fueron inspiradas al comprender el funcionamiento de los cerebros humanos y las interconexiones entre las neuronas. Pero, a diferencia de los cerebros humanos, donde cada neurona se puede comunicar con cualquier otra, las redes neuronales artificiales están compuestas por capas discretas, conexiones y direcciones de programación [6].

Una noción del funcionamiento de estas redes neuronales consiste en tomar una imagen y separarla en secciones que son introducidas en la primera capa de esta red. En esta primera capa, neuronas individuales ejecutan una tarea y transmiten esa información a la segunda capa. Esta segunda capa de neuronas realiza otro tipo de análisis sobre la imagen y así consecutivamente, hasta llegar a la última capa donde el resultado final es producido [6].

En la figura 2.1, se muestran los períodos por los cuales la inteligencia artificial ha evolucionado y se ha segmentado en las áreas de investigación de aprendizaje automático y aprendizaje profundo.

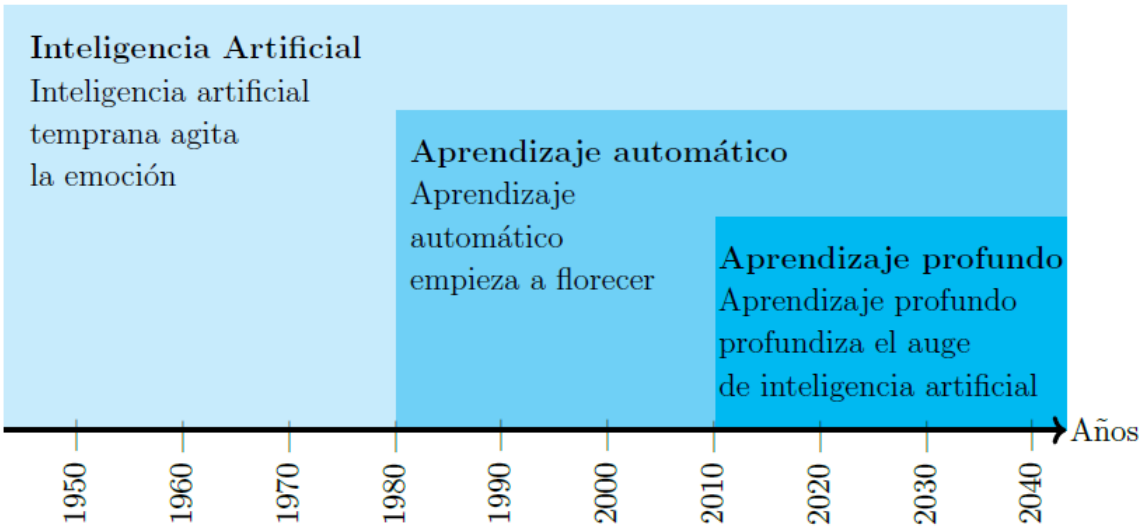


Figura 2.1: Evolución de inteligencia artificial [6].

En la siguiente sección se muestra una de las aplicaciones que tiene el aprendizaje profundo, las *CNN: Convolutional Neural Networks*, en español Redes Neuronales Convolucionales.

2.3 Redes Neuronales Convolucionales (CNN)

La clasificación de imágenes se refiere a la tarea de tomar una imagen y obtener como resultado una clase o etiqueta de algún objeto reconocido. Para los humanos, la tarea de reconocimiento de objetos es una de las primeras habilidades que aprenden desde el momento en que nacen. Por ejemplo, cuando se observa una escena, el ser humano es capaz de describirla y asignarle una etiqueta a cada objeto. Esta habilidad es la que se desea implementar mediante sistemas computacionales.

Un concepto muy importante de aclarar es la forma en que las máquinas "observan" las imágenes. Cuando una computadora observa una imagen, es decir, toma una imagen como entrada, no la observa como lo hacen los seres humanos, sino que interpretan esta imagen como un arreglo de valores de píxeles. Dependiendo de la resolución y tamaño de la imagen, las máquinas pueden observan un arreglo de tamaño 64 x 64 x 3, donde los valores de 64 se refieren a la altura y ancho de la imagen y el 3 se refiere a los canales de color.

Por dar un ejemplo, observe la imagen 640 x 640 x 1 en escala de grises mostrada en la figura 2.2, donde a cada uno de estos elementos, en esta matriz, se le asigna un valor en el rango de 0 a 255, el cual describe la intensidad del píxel. El objetivo final de las CNN es entregarle esta matriz a la computadora y que esta obtenga las probabilidades de encontrar objetos basados en esa información.

Al conocer los principios sobre cómo se interpretan y manipulan las imágenes, mediante

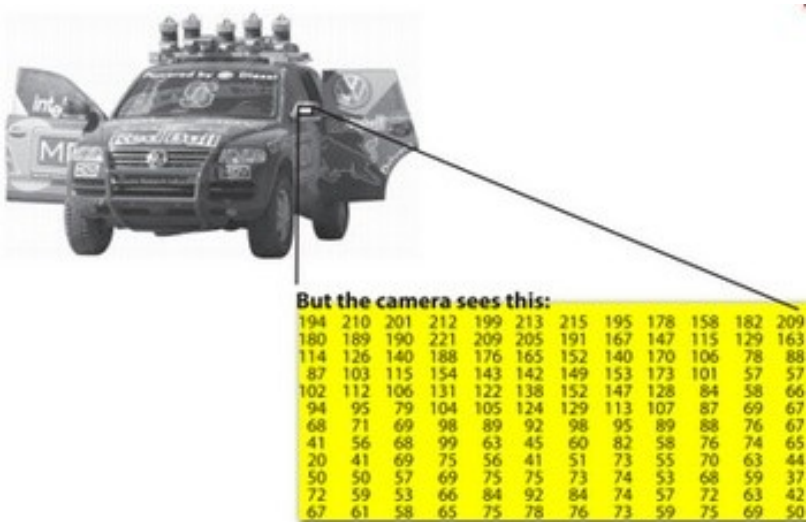


Figura 2.2: Como ven los seres humanos y las máquinas [7].

una computadora para reconocer objetos, esta debe lograr diferenciar entre todas las imágenes que se le brindan y descifrar las características únicas que hacen de un avión un avión o de una botella una botella. Este proceso se lleva a cabo en los seres humanos subconscientemente. Por ejemplo, cuando un ser humano ve a un caballo, este logra identificar que tiene cuatro patas y su característica cara y hocico alargados. Esto mismo es lo que se busca que logren las máquinas, identificar todos esos patrones que se logran desde un bajo nivel como curvas y bordes, hasta conceptos más abstractos que se logran con las redes neuronales convolucionales.

Las redes neuronales artificiales se inspiran en las redes neuronales biológicas. Por ejemplo, en el cerebro de los gatos, algunas neuronas se activan exclusivamente cuando observan bordes verticales y otras neuronas se activan exclusivamente cuando observan bordes horizontales [8]. Esto es lo que se procura conseguir con estas redes neuronales artificiales, que sean capaces de responder a características exclusivas de un objeto. Para lograr este objetivo, dichas redes se someten a una etapa de entrenamiento, donde aprenden habilidades para desempeñar una tarea en concreto, sin necesidad de una programación específica de tareas, de la misma forma en que los seres humanos desde pequeños aprenden a interpretar su entorno.

El modelo de las redes neuronales artificiales se compone de simples elementos llamados neuronas, como se muestra en la figura 2.3. La estructura básica de una neurona artificial está compuesta de n entradas, donde cada canal de entrada i puede transmitir un valor real x_i. La función de activación f calculada dentro de la neurona se selecciona de manera arbitraria. Usualmente, a los canales de entrada se les asigna un peso, lo cual significa que la información de entrada es multiplicada por estos pesos w_i. La información transmitida se introduce en la neurona y la función de activación es evaluada [9].

El modelo matemático de cada neurona se puede pensar como un clasificador lineal, lo cual se refiere a que una neurona tiene la capacidad de que le "gusten" (activación cerca de 1) o "no le gusten" (activación cerca de 0) ciertas regiones del espacio de entrada. Por

lo tanto, con una apropiada función de pérdida (en inglés *loss function*) en la salida de la neurona, se puede convertir a una neurona individual en un clasificador lineal. Algunas de estas funciones son: *Binary Softmax classifier, Binary SVM classifier, Regularization interpretation*, entre otras.

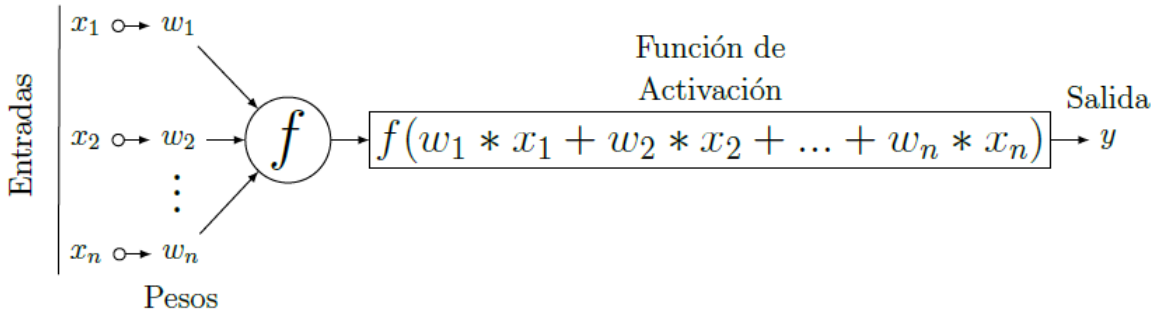


Figura 2.3: Estructura básica de una neurona artificial [9].

Si se acepta el hecho de que cada función de activación es capaz de tomar las entradas, procesarlas y luego mostrar una salida precisa, entonces se asume que las redes neuronales artificiales son, en esencia, redes de funciones de activación. Distintos modelos de redes neuronales artificiales se diferencian por razones como la función de activación utilizada, los patrones de interconexión y el tiempo que tarda transmitir información [9].

Seguidamente, se muestra en la figura 2.4 una estructura de una red neuronal, la cual está compuesta por una capa de entrada, una capa de salida, y puede contener una o varias capas ocultas o intermedias [9]. Esta red recibe una entrada (un vector simple) y la transforma a través de una serie de capas ocultas. Cada una de estas capas ocultas está compuesta por un conjunto de neuronas, donde cada neurona está completamente conectada a todas las neuronas de la capa anterior y donde neuronas de una misma capa son independientes y no comparten conexión.

Los tres elementos principales que definen el comportamiento de una red neuronal artificial son la estructura de los nodos, la topología de la red y el algoritmo de aprendizaje utilizado para encontrar los pesos de la red [9].

Para continuar con los detalles específicos de una red del tipo CNN, la idea es que la red reciba una imagen como entrada e introducirla en distintas capas de convolución, capas de funciones de activación, *pooling* y *fully connected*, y se obtenga como resultado una clase o probabilidad de haber reconocido un objeto en la imagen. En los siguientes párrafos, se detalla en qué consisten estas cuatro principales capas mencionadas previamente.

La capa de convolución consiste en un conjunto de filtros que llegan a ser aprendidos por la red. Cada filtro es pequeño, especialmente en las dimensiones de altura y ancho, pero se extiende a lo largo de toda la profundidad de los datos de entrada, con distintos pesos. Por ejemplo, un filtro común podría tener un tamaño de 5x5x3, lo cual se refiere a cinco píxeles de altura, cinco píxeles de ancho y tres píxeles de profundidad referentes a los canales de color. Estos filtros se deslizan o convolucionan en las direcciones de altura y

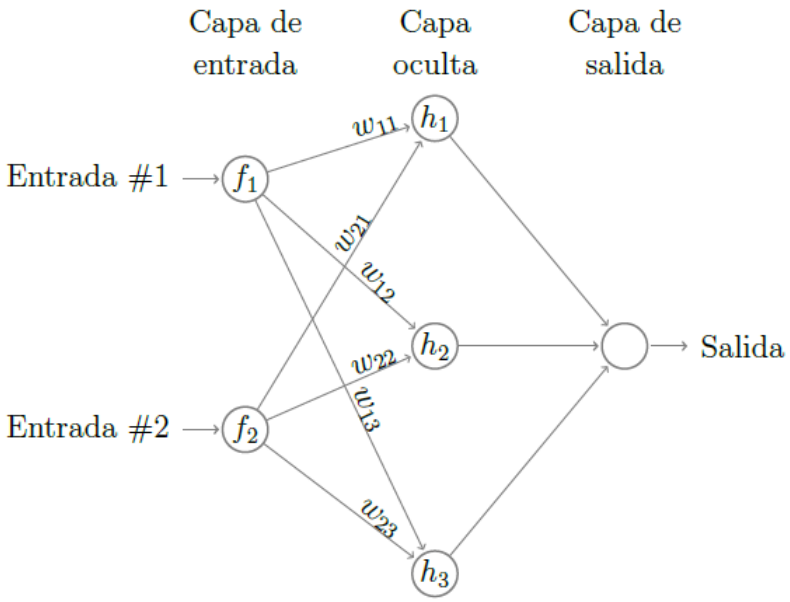


Figura 2.4: Estructura básica de una red neuronal.

ancho a lo largo de toda la información de entrada, como se muestra en la figura 2.5, en este caso una imagen, y se multiplica uno a uno los valores del filtro y cualquier posición en la imagen, como se muestra en la figura 2.6 (el cálculo para esta imagen se muestra en la ecuación 2.1). Este proceso se repite en cada posición de la imagen, en las direcciones de altura y ancho, hasta conseguir la capa convolucionada, también denominada mapa de activación.

$$\begin{aligned} \textit{Capa Entrada} \cdot \textit{Filtro} = {} & 9*1+0*0+9*0+0*0+3*1+ \\ & 8*0+1*1+1*0+2*1+2*0+ \\ & 1*0+7*0+7*1+8*0+8*0+ \\ & 6*0+4*1+8*0+2*1+6*0+ \\ & 4*1+2*0+0*0+0*0+5*1 \\ & \therefore \textit{Resultado de Convolucion} = \mathbf{37} \end{aligned} \tag{2.1}$$

Conforme se convoluciona, este proceso produce un mapa de activación de dos dimensiones, el cual muestra la respuesta de un filtro específico en cada posición espacial, en la información de entrada. Intuitivamente, la red aprenderá filtros con sus respectivos pesos, que se activan cuando detectan alguna característica visual como, por ejemplo, un borde con una orientación característica o una mancha de algún color en la primera capa (bajo nivel) e incluso un objeto entero en capas de más alto nivel. De esta forma, si se tienen 12 filtros, cada uno de estos mapas de activación se apilarán a lo largo de la dimensión de profundidad y se producirá el volumen de salida.

La capa de funciones de activación toma un número individual y lo procesa a través de una operación matemática fija. Existen varios tipos de funciones de activación que se

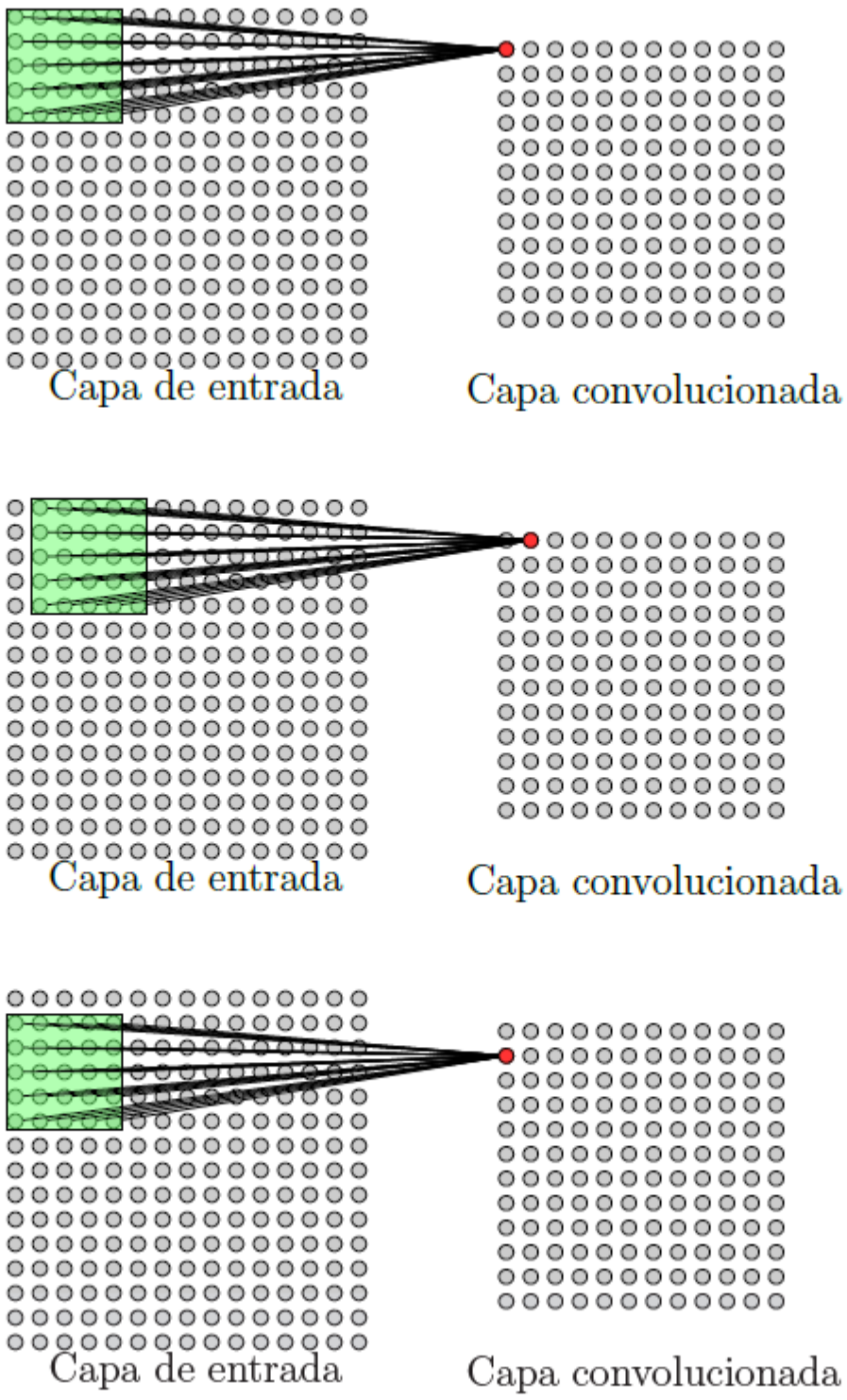


Figura 2.5: Proceso de convolución con un filtro 5x5 (Cuadro).

encuentran en la práctica, como Sigmoide, tangente hiperbólica, RELU (*Rectifier Linear Unit*), entre otras [10]. Otra capa importante es la capa de *pooling*, donde su frecuencia de uso es alta entre capas de una arquitectura CNN. Su función, como se muestra en

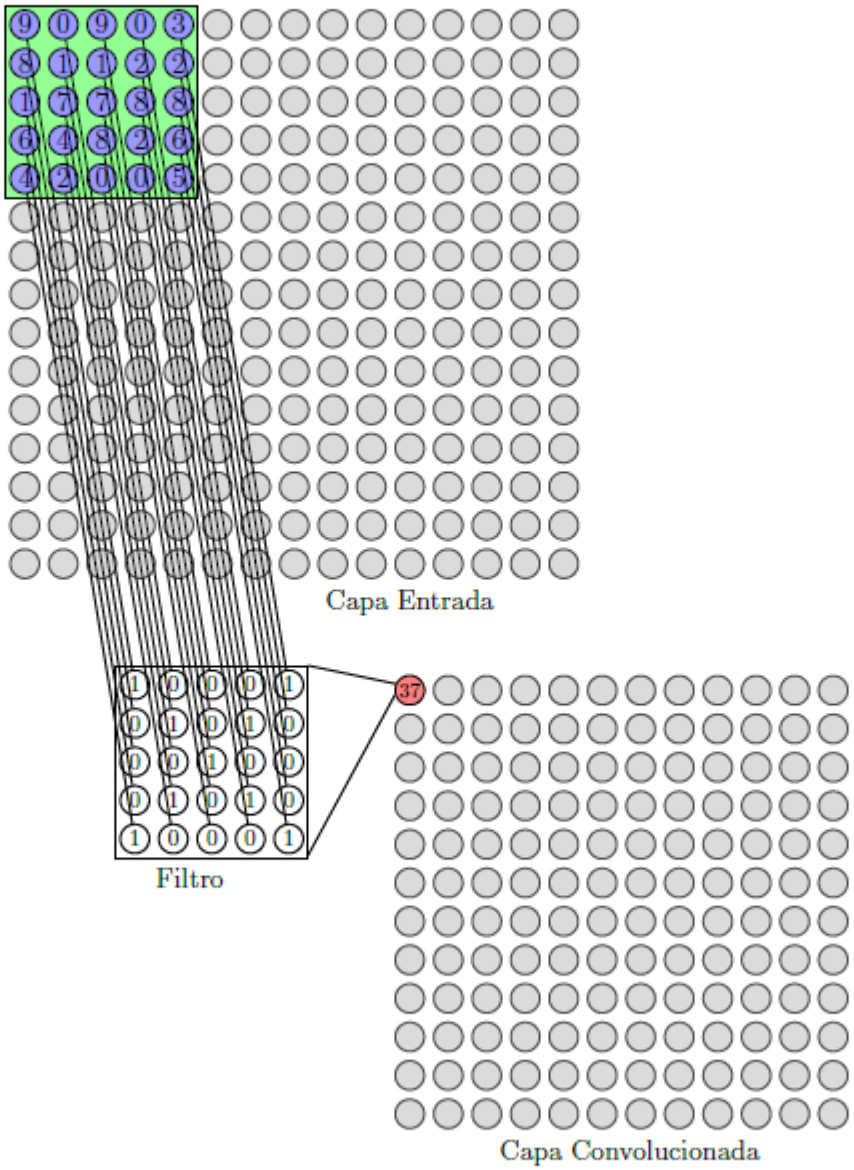


Figura 2.6: Cálculo al aplicar un filtro de convolución.

la figura 2.7, es reducir progresivamente el tamaño del espacio de alguna representación brindada por una capa, con el fin de disminuir la cantidad de parámetros y cálculos en la red, pero también buscar características de más alto nivel.

Por último, la capa *fully connected* toma como entrada el volumen de salida de la capa anterior (ya sea una capa de convolución, de función de activación, *pooling*) y produce un vector de dimensión N, donde N es el número de clases que calcula la red. Por ejemplo, si se desea un programa de clasificación de dígitos manuscritos entrenado con el conjunto de datos MNIST [11], N respondería a los 10 dígitos en el sistema decimal. Cada número en este vector de dimensión N representa la probabilidad de haber reconocido cierta clase (en este caso, la probabilidad de haber reconocido un dígito donde la primera clase es el número cero y, mediante incrementos unitarios, llegar a la última clase que sería el número nueve). En un caso hipotético de que el resultado de esta capa *fully connected*

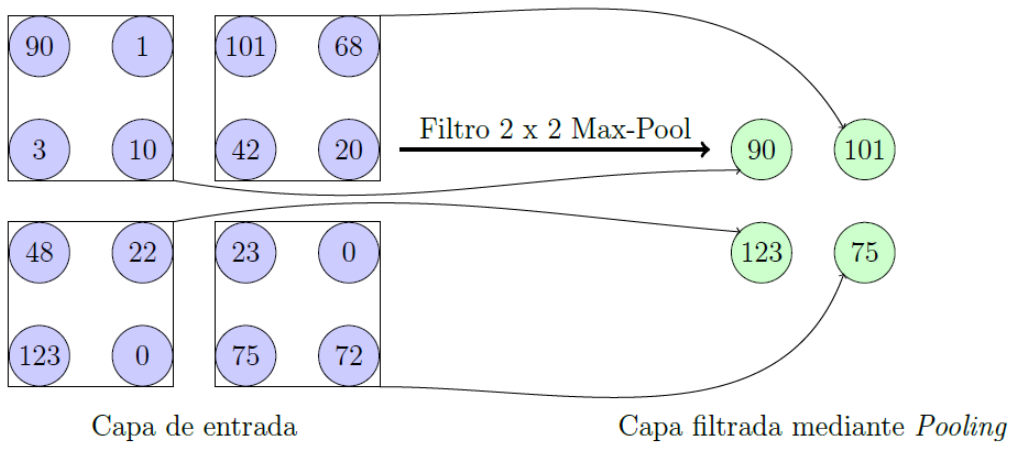


Figura 2.7: Capa *pooling*, con filtro 2x2 y función máximo.

sea el de la figura 2.8, representado por el vector de la ecuación 2.2, se puede concluir que en la imagen de entrada se reconoció el número 5 con una probabilidad de 0.75.

$$N = [0 \quad 0.1 \quad 0 \quad 0 \quad 0.1 \quad 0.75 \quad 0 \quad 0 \quad 0 \quad 0.05] \tag{2.2}$$

La capa RELU aplicará una función de activación, lo cual deja el tamaño del volumen sin cambios (32x32x12). Luego, la capa de *pooling* realiza un proceso llamado *downsampling* que se encarga de reducir el tamaño de la matriz de trabajo, por ejemplo, a 16x16x12. Por último, la capa *fully connected* calcula las probabilidades para cada clase, lo cual resulta en una matriz 1x1x10, donde cada uno de esos 10 números corresponden a las probabilidades de las clases, que directamente son las 10 categorías del conjunto de datos CIFAR-10 [12].

Con lo mencionado hasta el momento, es posible identificar objetos dentro de una imagen, pero no determinar la posición en que están localizados, por este motivo, en el siguiente apartado, se muestra una aproximación de cómo lograrlo con las *RPN: Region Proposal Networks*, que en español se puede traducir como Redes de Propuestas de Regiones.

2.4 RPN: Region Proposal Networks

Las RPN tienen como objetivo buscar zonas dentro de una imagen donde existe la posibilidad de encontrar objetos. Para explicar cómo funciona la arquitectura de las RPN mostradas en la figura 2.11, se utilizará el ejemplo de la figura 2.12.

Cuando se quiere buscar un objeto dentro de una imagen de entrada con este método, lo primero es aplicar una serie de capas convolucionales (en este ejemplo el modelo VGG-16 [13], para obtener un *feature map* o mapa de características. Para generar estas propuestas de regiones, se desliza una pequeña red a lo largo del mapa de características generado por

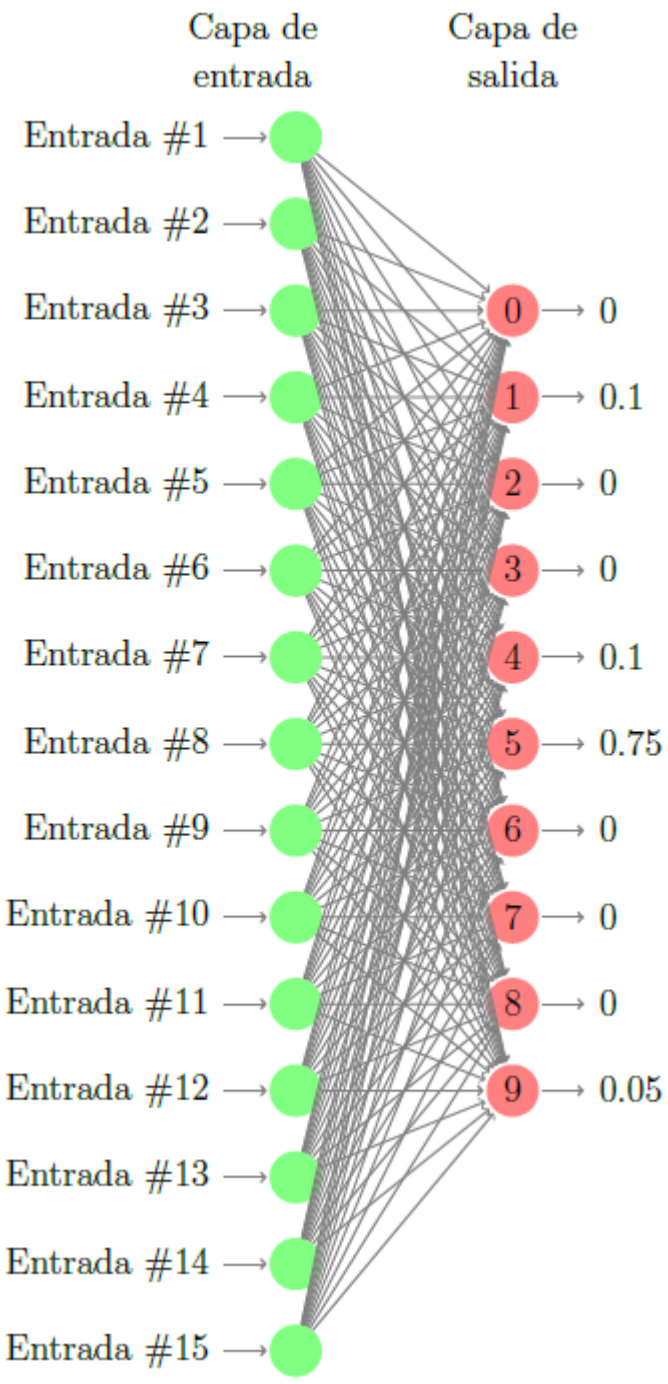


Figura 2.8: Capa *fully connected*.

la última capa convolucional del bloque VGG-16. Esta pequeña red toma como entrada una ventana de dimensiones nxn píxeles de la entrada que se refiere al *feature map*. Este último *feature map* es alimentado a dos capas fully connected, una capa *box regression (reg)* y a una *box-classification (cls)*. Para este ejemplo se utiliza n=3.

En cada posición de esta ventana deslizante, de forma simultánea, se predicen múltiples propuestas de regiones, donde el número máximo de propuestas de regiones para cada posición se denota k. De esta manera, la salida de la RPN está compuesta por la capa

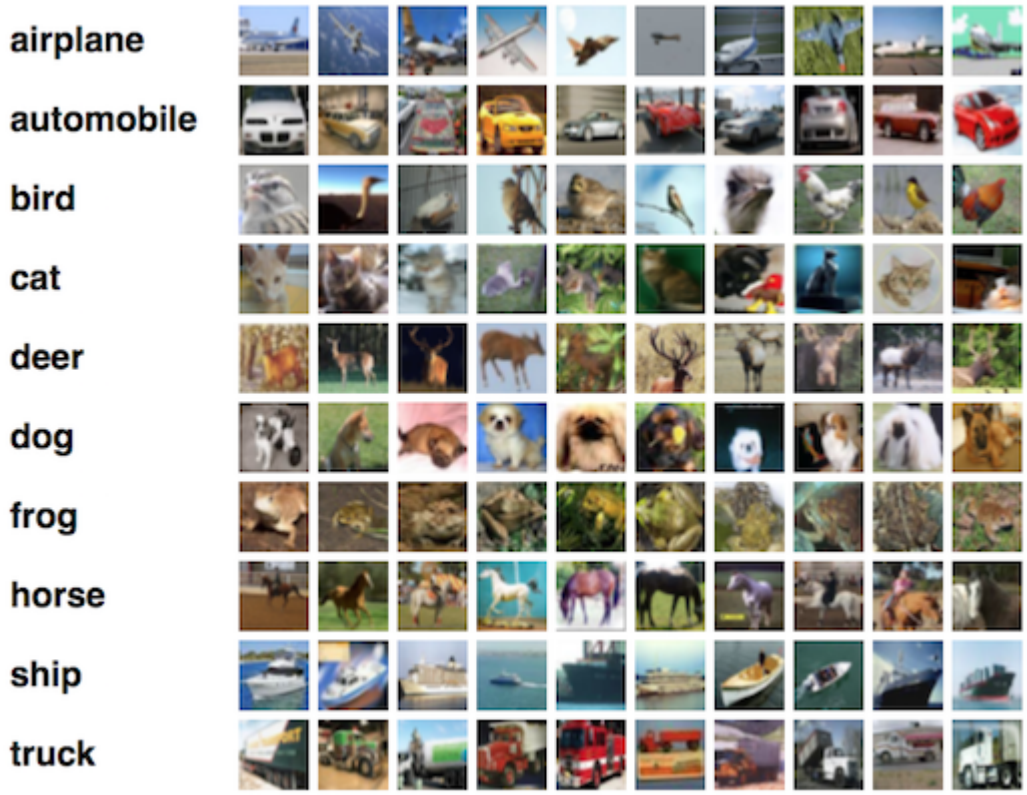


Figura 2.9: Conjunto de datos CIFAR-10 [12].

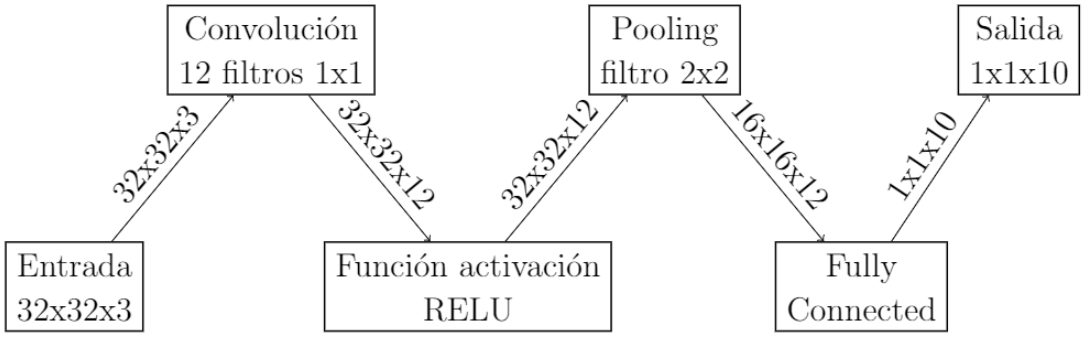


Figura 2.10: Arquitectura CNN para CIFAR-10.

Box-regression, la cual tiene $4k$ salidas que se refieren a las coordenadas de k regiones de interés y la capa *Box-classification* con $2k$ puntajes que estiman la probabilidad de detectar un objeto o no, tal como se muestra en la figura 2.13. A las propuestas k se les denomina anclas, donde cada una está centrada en la ventana deslizante en cuestión y tiene asociados una escala y razón de aspecto. Para un *feature map* de convolución convencional de tamaño W x H (2400), existen WHk anclas en total.

Esta pequeña red se muestra en la esquina superior izquierda de la figura 2.11. En la salida de la RPN, se obtiene un conjunto de propuestas rectangulares de objetos, conocidas como regiones de interés (en inglés ROI: *Regions Of Interest*) y dos probabilidades. De esta manera, la salida está compuesta por seis parámetros, los dos primeros: *Es objeto y No es*

objeto, se refieren a la probabilidad de que exista un objeto en la región propuesta, tienen esta configuración, ya que, al ser un parámetro que debe utilizarse en otra red, se utilizan en el formato *one-hot-encoding*; los otros cuatro parámetros se refieren a las regiones de interés, compuestas por las dimensiones del rectángulo y las coordenadas donde se encuentra su centro, como se muestra en la figura 2.14 [14].

Las RPN han sido diseñadas para predecir propuestas de regiones con un rango amplio de escalas y relaciones de aspecto, en contraste con los métodos convencionales que recurrían a métodos de pirámides de imágenes o pirámides de filtros [15] [16] [17].

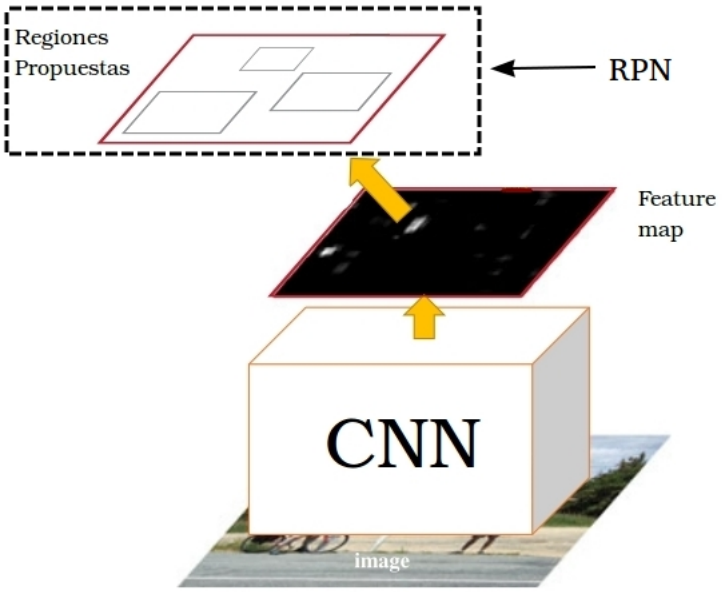


Figura 2.11: Arquitectura de RPN.

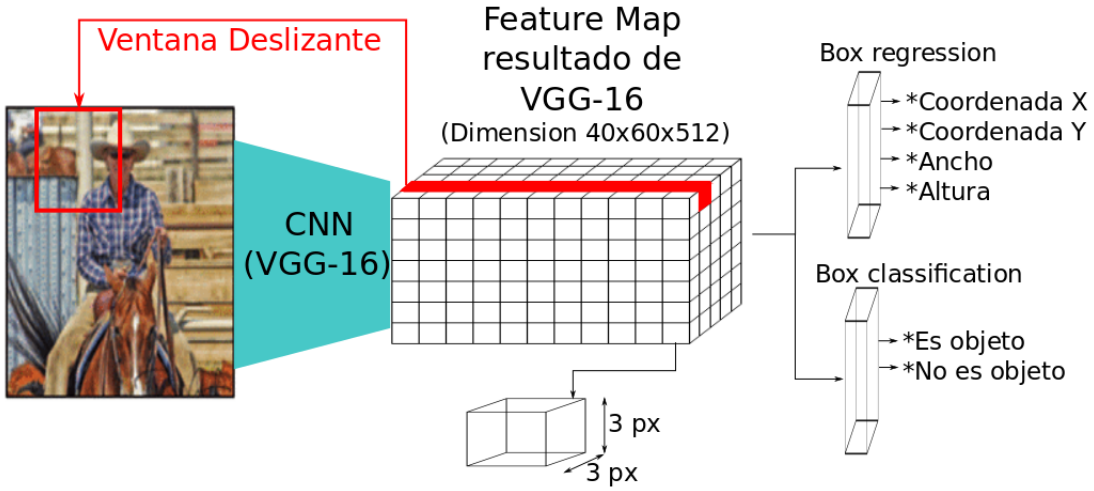


Figura 2.12: Ejemplo de arquitectura RPN.

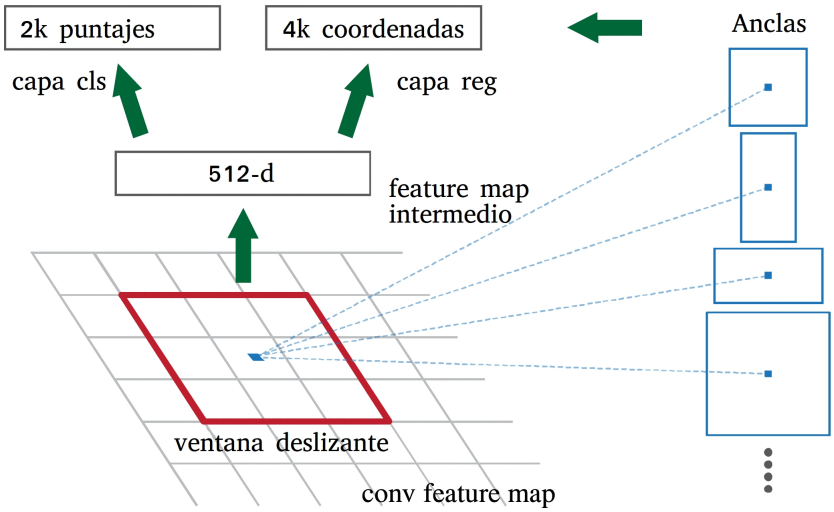


Figura 2.13: Ventana deslizante de RPN.

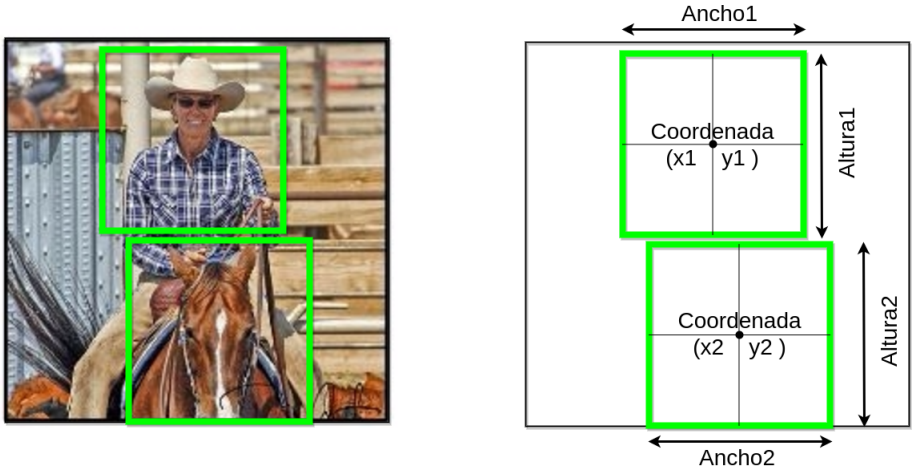


Figura 2.14: Regiones de interés generadas por RPN.

2.5 R-CNN: Region Convolutional Neural Networks

Al igual que la anterior RPN, las redes del tipo R-CNN tienen la función de tomar como entrada una imagen y en ella detectar, pero, además, clasificar objetos. Existen varias arquitecturas para este tipo de redes, a continuación, se muestra la R-CNN que se apega a este proyecto, una arquitectura estructurada en tres etapas, tal y como se muestra en la figura 2.15. La primera etapa genera propuestas de regiones independientemente de la categoría del posible objeto. En la segunda etapa, se extrae un vector de longitud fija con características de cada región propuesta por la primera etapa. El tercer módulo es un *class-specific linear Support Vector Machine* [18].

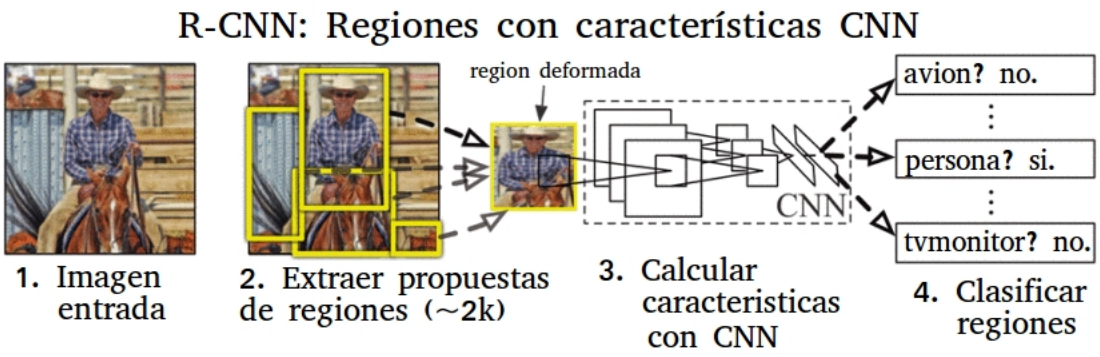


Figura 2.15: Arquitectura de red R-CNN.

2.5.1 Reconocimiento utilizando regiones

Este algoritmo funciona para la detección de objetos [19], el cual inicia con la construcción de un árbol utilizando un motor de segmentación jerárquica [20]. Las regiones que se toman en consideración son los nodos de ese árbol y la raíz, la cual es la imagen completa para lograr esta aproximación. En la figura 2.16, se presenta un ejemplo de un árbol de regiones, junto con un saco de regiones que representan la imagen de entrada, donde este grupo descarta la estructura del árbol. Luego, para describir una región, se subdivide el *bounding box* (este término se refiere a la región de interés) en una malla nxn, tal como se muestra en la figura 2.17.

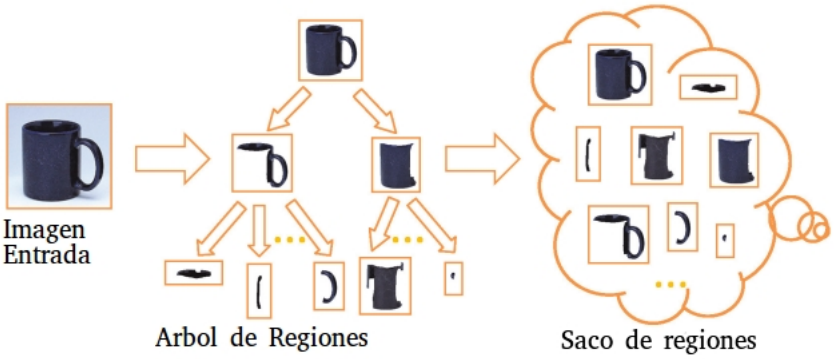


Figura 2.16: Extracción de regiones [21].

El flujo de este sistema de reconocimiento consiste en tres etapas, como se muestra en la figura 2.18. Para una imagen de consulta, la etapa de votación emite hipótesis iniciales de posiciones de objetos, escalas y soportes basados en regiones combinadas de ejemplos. Estas hipótesis son las entradas a las siguientes etapas y son refinadas a través de un clasificador de verificación y una segmentación restringida, respectivamente, para obtener la detección final y resultados de segmentación.

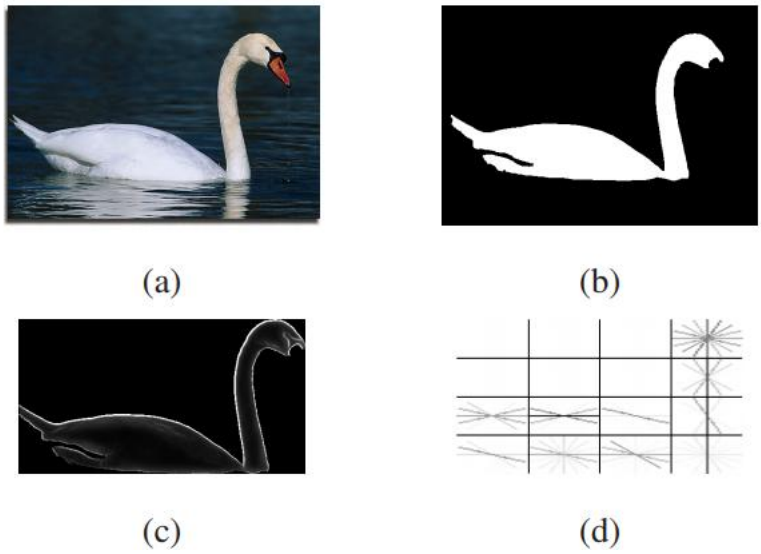


Figura 2.17: Descripción de regiones [21].

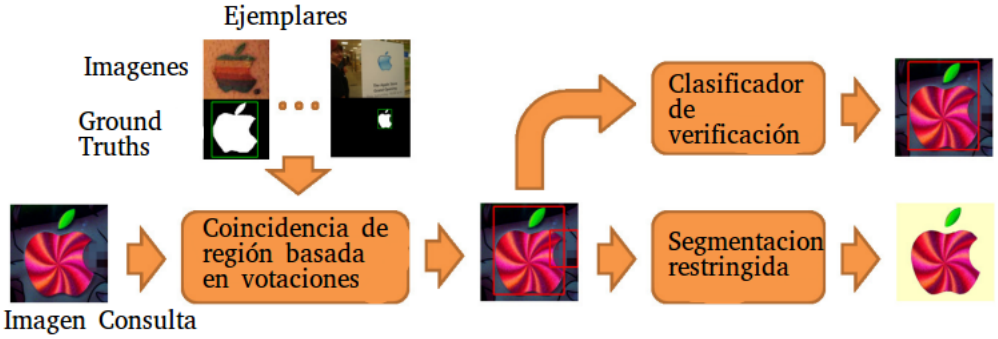


Figura 2.18: Diagrama de flujo en el reconocimiento utilizando regiones [21].

2.5.2 Extracción de características

En esta etapa, se utilizan las regiones propuestas en la sección 2.5.1, las cuales son deformadas hasta lograr una dimensión de 227 x 227 píxeles, debido a que se deben apegar a las condiciones de entrada de la red CNN. Esta etapa de procesamiento de la imagen está compuesta por cinco capas de convolución y dos capas *fully connected*. En la salida, se obtiene un vector de características de 4096 dimensiones de cada una de las propuestas de región, las cuales se calculan aplicando *forward propagation*.

2.5.3 Class-specific linear support vector machine

El aprendizaje automático del tipo *support vector machine* (máquinas de vectores de soporte) son modelos de aprendizaje supervisado con algoritmos de clasificación basados en regresiones. Dado un conjunto de ejemplos de entrenamiento, cada uno etiquetado

en la categoría que pertenece, en un conjunto de dos o más categorías, un algoritmo de entrenamiento SVM construye un modelo que posee un comportamiento de un clasificador lineal binario no probabilístico.

Para tener una idea más clara de cómo funciona este algoritmo, se debe imaginar alguna forma de clasificar, mediante procesamiento de imágenes, una pelota y una silla. El primer paso es enseñarle a este algoritmo a diferenciar entre estos dos objetos. Esto lo logra utilizando n características de los objetos analizados, en este ejemplo n=2, altura y ancho del objeto. Se debe suponer que la relación de aspecto de estos objetos tiene una estructura ancho: altura, donde para una pelota es aproximadamente 1:1 (figura 2.19) y para una silla 2:3 (figura 2.20). Este aprendizaje se logra cuando el modelo observa un conjunto de datos con información relevante de los objetos, en este caso, se utilizará una etiqueta que identifique al objeto, su altura y su ancho, como se muestra en la tabla 2.1.

Figura 2.19: Relación de aspecto de una pelota.

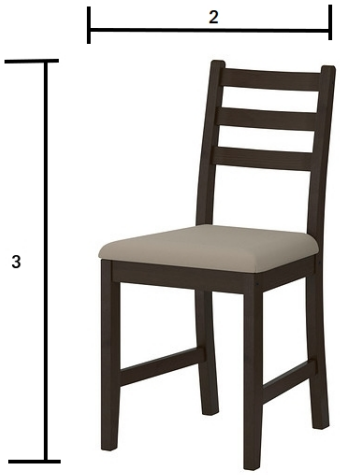


Figura 2.20: Relación de aspecto de una silla.

Etiqueta de objeto	Altura (píxeles)	Ancho (píxeles)	Altura normalizado (adimensional)	Ancho normalizada (adimensional)
pelota1	15	16	0.1630434783	0.1739130435
pelota2	30	29	0.3260869565	0.3152173913
pelota3	70	75	0.7608695652	0.8152173913
pelota4	35	33	0.3804347826	0.3586956522
pelota5	89	92	0.9673913043	1
silla1	29	47	0.3152173913	0.5108695652
silla2	45	70	0.4891304348	0.7608695652
silla3	10	18	0.1086956522	0.1956521739
silla4	59	87	0.6413043478	0.9456521739
silla5	20	31	0.2173913043	0.3369565217

Tabla 2.1: Altura y ancho de objetos de entrenamiento para SVM.

Observe que, para utilizar estos datos, un paso crítico es normalizar la información, con el fin de que tome sentido relevante entre los objetos analizados. Lo siguiente es construir un histograma, a partir de este conjunto de características normalizadas, para determinar cuáles patrones diferencian a estos objetos. Si se observa detenidamente el histograma de la figura 2.21, se puede inferir que las barras de altura y anchura de una pelota son similares, mientras que, para las sillas, difieren a razón de que la altura es mayor al ancho, lo cual comprueba la hipótesis inicial sobre la razón de aspecto.

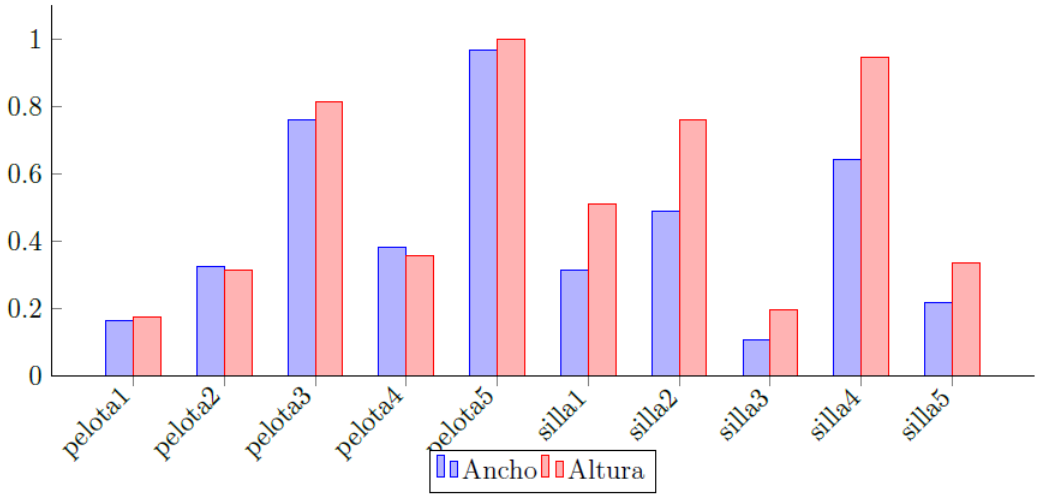


Figura 2.21: Histograma para ejemplo de SVM.

Por lo tanto, la forma en que aprende el algoritmo de SVM a diferenciar objetos es determinar un hiperplano (líneas en la figura 2.22) que logre separar estos dos grupos de puntos en este gráfico de dispersión. Es fácil pensar que existen muchas líneas que

pueden separar estas dos aglomeraciones de objetos, pero las que garantizan los mejores resultados son las líneas que permitan un margen máximo entre los elementos de las dos categorías (línea continua) y las demás propuestas se descartan (línea punteada y línea discontinua).

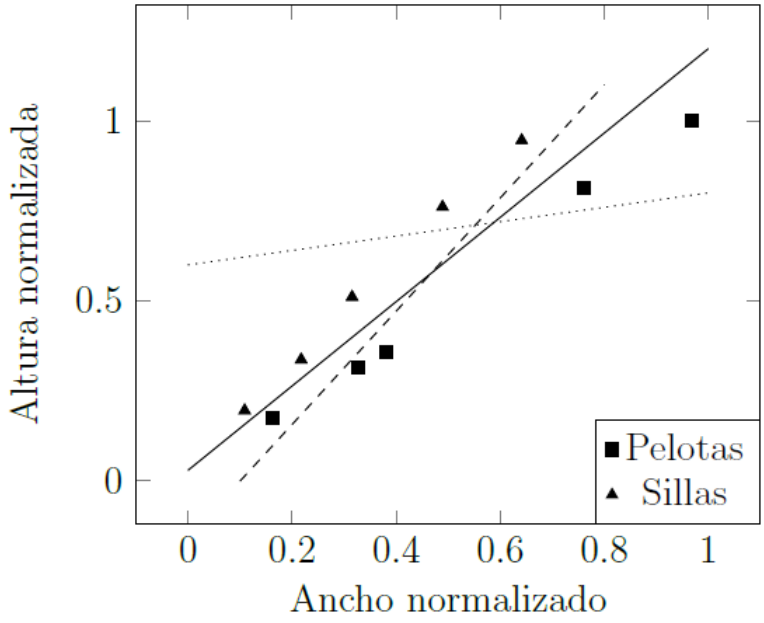


Figura 2.22: Propuestas de hiperplanos para SVM.

Sería ideal que, con solo estas dos características, altura y ancho, se pudieran reconocer todos los objetos, pero no es así. Por esta razón, a lo largo de la R-CNN, se pueden obtener muchas características, para este caso n=4096. Cuando se construye un histograma para cada objeto, cada uno de estos tendrá un patrón característico, por esta razón, el modelo SVM busca similitudes entre los histogramas de los objetos aprendidos y una imagen de entrada. Un ejemplo de un histograma de 4096 características de un objeto aprendido se observa en la figura 2.23.

De esta manera, después de haber entrenado un modelo SVM, este podrá predecir que, si un objeto coincide con estos patrones característicos de los objetos aprendidos, tendrá altas probabilidades de haber sido reconocido.

2.6 Faster R-CNN

El modelo Faster R-CNN es una fusión de las redes RPN y R-CNN; el término *faster* (en español se traduce como "más rápido") se refiere a que esta es una de las optimizaciones más eficientes en el tema de velocidad de detección que se le ha hecho a una serie de modelos basados en la arquitectura de la R-CNN. En la figura 2.24, se puede observar, al lado izquierdo, la RPN, la cual propone las regiones de interés (ROI) y, en la parte superior, se puede observar el clasificador R-CNN. En pocas palabras, después de aplicar

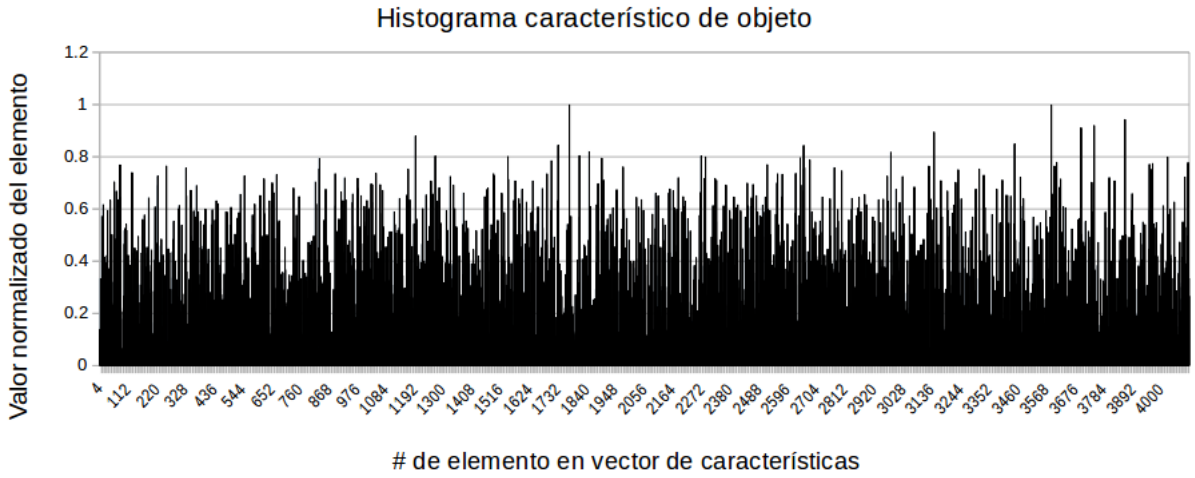


Figura 2.23: Histograma característico de un objeto aleatorio para SVM.

una red convolucional (para efectos de estos proyectos, esta implementación se basa en el modelo VGG-16 antes mencionado), a la imagen en cuestión, por ejemplo, a la imagen de la figura 2.25, la RPN le dice dónde buscar a la R-CNN. El objetivo de esta implementación es sustituir la etapa de reconocimiento utilizando regiones en la R-CNN convencional por la RPN, ya que esta última es más rápida calculando las propuestas de región y, consecuentemente, la R-CNN cumple su función como clasificador de objetos.

La salida de la Faster R-CNN está compuesta por la clase del objeto, la probabilidad de haber identificado el objeto y las cuatro esquinas del *bounding box* que localiza el objeto. Por ejemplo, si se utiliza una red Faster R-CNN entrenada con el conjunto de datos Pascal VOC 2012 [22] que tiene 20 categorías, para una imagen de entrada mostrada en la figura 2.25, los resultados de detección de objetos se muestran en la figura 2.26 para la categoría perro y en la figura 2.27 para la categoría persona.

2.7 Conjunto de datos

El conjunto de datos se refiere a toda la información, que será utilizada para las etapas de entrenamiento y prueba de la red neuronal. La manera en que se prepara consiste en, primero, recolectar múltiples imágenes (idealmente mayor a 5000 imágenes) de los objetos que se pretenden reconocer. Una forma eficiente de realizarlo es tomar un video del objeto desde diferentes perspectivas, en diferentes entornos y luego extraer todos los cuadros del video. Una recomendación para tomar el video es utilizar una cámara con estabilizador de imagen, si esto no se cumple, las imágenes del conjunto de datos contendrán ruido que afectará la etapa de entrenamiento.

En el segundo paso, por facilidad, se deben agrupar todas las imágenes de una sola categoría en una carpeta. Luego, de esas 5000 imágenes, se realiza una selección aleatoria

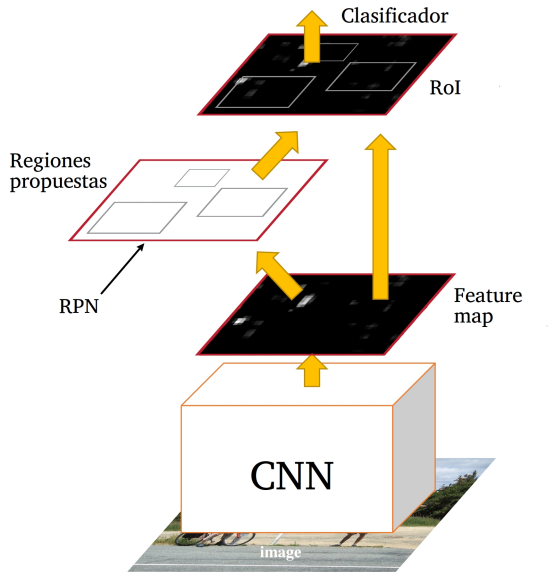


Figura 2.24: Arquitectura de Faster R-CNN.

Figura 2.25: Imagen de entrada a Faster R-CNN.

de 1000 imágenes. Con base en estas 1000 imágenes, se crea un *manifest file*, el cual se define como un archivo que contiene metadatos para un grupo de archivos adjuntos, los cuales forman parte de un conjunto o unidad coherente. En otras palabras, es un archivo que contiene la dirección que apunta a cada una de estas imágenes acompañado del *ground truth* compuesto por la posición en la lista de categorías de salida de la red y el *bounding box* donde se encuentra el objeto, donde primero se indica su esquina superior izquierda y luego la esquina inferior derecha. Por ejemplo, basado en la imagen mostrada en la figura 2.28, una línea de este *manifest file* tiene un aspecto similar a lo que se muestra en la figura 2.29.

Para la imagen de la figura 2.28, una línea real sería algo similar a la siguiente figura 2.30:

Después de construir este *manifest file*, que contenga alrededor de 1000 líneas similares a estas anteriores, continúa la etapa de entrenamiento de la red.

Figura 2.26: Salida de Faster R-CNN: categoría perro.

Figura 2.27: Salida de Faster R-CNN: categoría persona.

2.8 Entrenamiento de una red neuronal artificial

Para entrenar un modelo específico, no es necesario entrenar el modelo desde cero, a menos que se tenga un conjunto de datos extremadamente grande, por ejemplo, el conjunto de datos ImageNet [23]. De lo contrario, se puede entrenar un modelo específico a partir de un modelo Faster R-CNN pre-entrenado, mediante la técnica *fine tuning*, el cual es el proceso en el que los parámetros de un modelo deben ajustarse con precisión. La razón es porque una R-CNN pre-entrenada contiene muchas características buenas de bajo nivel, que se pueden usar de forma general. Incluso, si se pretende utilizar una nueva arquitectura (es decir, sin un modelo Faster R-CNN preentrenado existente), es recomendable seguir el método de entrenamiento de Faster R-CNN, donde primero se entrena la Faster R-CNN mediante *backpropagation* y luego, se aplica un *fine tunning* utilizando el conjunto de datos propio [24].

Figura 2.28: Imagen llamada botella.jpg donde, adicionalmente, se muestra su *bounding box*.

```
direccion_del_archivo/nombreImagen.jpg,bbox1[x],bbox1[y],
   bbox2[x],bbox2[y],numero_en_lista_categorias
```

Figura 2.29: Línea en el *manifest file* que describe al objeto dentro de la imagen.

2.9 Robot Pepper

Como se mencionó en el capítulo 1, Pepper es un robot humanoide con gran cantidad de sensores y actuadores, mostrados en el anexo A.1. En la figura 2.31, se muestran los nombres de todas las articulaciones del robot Pepper.

Un aspecto importante en el campo de la robótica es conocer los rangos en los cuales se pueden mover las articulaciones. Tomando como referencia un sistema de coordenadas, también conocido como trama [25], mostrado en la figura 2.32. Se muestran estos rangos en la figura 2.33 para la cabeza del robot y en la figura 2.34 para el brazo izquierdo del robot que, además, pueden ser reflejados en el brazo derecho. Toda esta información está presente en la documentación de Pepper, en el sitio web de SoftBanks Robotics [26].

2.9.1 Articulaciones de la cabeza

Los rangos de movimiento de las articulaciones presentes en la cabeza del robot se muestran en la tabla 2.2 y figura 2.33. Debido a la colisión con la carcasa del robot y la tableta, los movimientos alrededor de HeadYaw y HeadPitch de la cabeza están limitados, tal como se muestra en la tabla 2.3.

```
direccion_del_archivo/botella.jpg,45,30,270,580,1
```

Figura 2.30: Línea que describe a la botella dentro de la imagen de la figura 2.29.

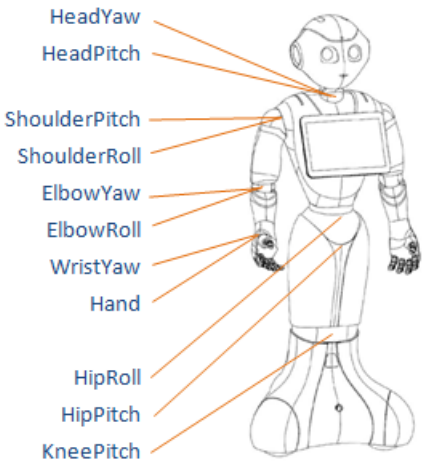


Figura 2.31: Articulaciones de Pepper.

Nombre de articulación	Movimiento (Eje de rotación)	Rango (°)	Rango (rad)
HeadYaw	Articulación cabeza giro (Z)	-119.5 a 119.5	-2.0857 a 2.0857
HeadPitch	Articulación cabeza atrás y adelante (Y)	-40.5 a 36.5	-0.7068 a 0.6371

Tabla 2.2: Rangos de movimientos de articulaciones de la cabeza del robot Pepper.

Restricción	HeadYaw (°)	HeadPitch (°)	HeadPitch (°)
1	-119.5	-35.0	13.5
2	-91.4	-35.1	13.5
3	-61.6	-35.2	20.9
4	-33.33	-40.5	36.5
5	33.33	-40.5	36.5
6	61.6	-35.2	20.9
7	91.4	-35.1	13.5
8	119.5	-35.0	13.5

Tabla 2.3: Restricción de movimientos de articulaciones de la cabeza del robot Pepper.

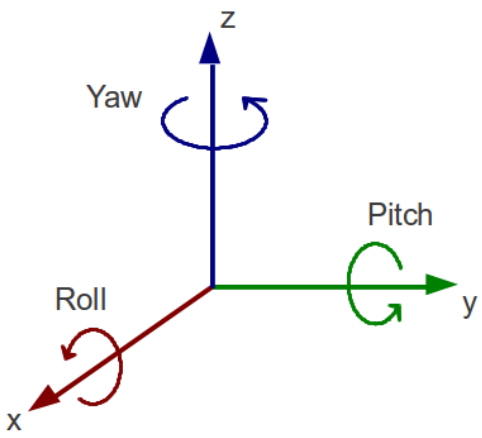


Figura 2.32: Trama de referencia de Pepper.

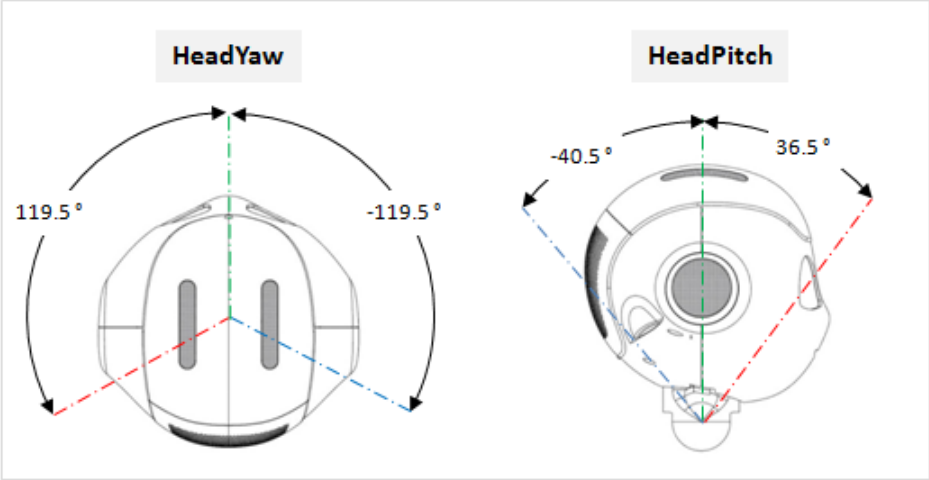


Figura 2.33: Diagrama de rangos de movimiento de la cabeza del robot Pepper.

2.9.2 Actuadores y articulaciones del brazo izquierdo

Los rangos de movimiento de las articulaciones presentes en el brazo izquierdo del robot se muestran en la tabla 2.4 y figura 2.34. Para evitar colisiones, la articulación LElbowRoll está limitada en los ángulos de movimiento, esta información se muestra en la tabla 2.5.

2.10 ROS: Robot operating system

El responsable de lograr la integración de todas estas herramientas, para que funcionen de manera organizada, es el sistema operativo de robots ROS (en inglés *Robot Operating System*), el cual es un *software* enfocado en robótica de tipo *middleware*. ROS provee servicios diseñados para clústeres heterogéneos de computadoras, tales como abstracción

Nombre de articulación	Movimiento (Eje de rotación)	Rango (°)	Rango (rad)
LShoulderPitch	Left shoulder joint front and back (Y)	-119.5 a 119.5	-2.0857 to 2.0857
LShoulderRoll	Left shoulder joint right and left (Z)	0.5 a 89.5	0.0087 to 1.5620
LElbowYaw	Left shoulder joint twist (X')	-119.5 a 119.5	-2.0857 to 2.0857
LElbowRoll	Left elbow joint (Z')	-89.5 a -0.5	-1.5620 to -0.0087
LWristYaw	Left wrist joint (X')	-104.5 a 104.5	-1.8239 to 1.8239

Tabla 2.4: Rango de movimientos de actuadores y articulaciones del brazo izquierdo del robot Pepper.

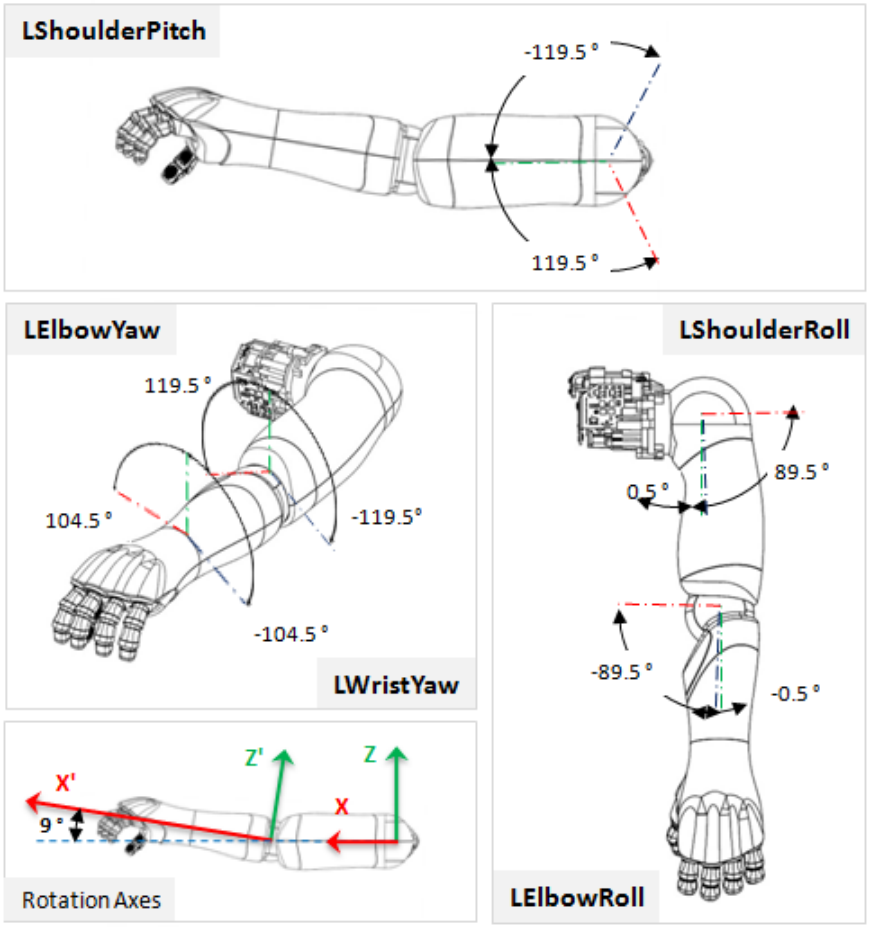


Figura 2.34: Diagrama de rangos de movimientos del brazo izquierdo del robot Pepper.

de *hardware*, control de dispositivos de bajo nivel, implementación de funcionalidades de uso común, paso de mensajes entre procesos y administración de paquetes. También proporciona herramientas y librerías para obtener, compilar, escribir y ejecutar código a través de múltiples ordenadores [27].

Restricción	LElbowYaw (°)	LElbowRoll Min (°)	LElbowRoll Max (°)
1	-119.5	-78.0	-0.5
2	-60.0	-78.0	-0.5
3	0.0	-89.5	-0.5
4	99.5	-89.5	-0.5
5	119.5	-83.0	-0.5

Tabla 2.5: Restricción de movimientos de articulaciones del brazo izquierdo del robot Pepper.

Los conceptos importantes que se deben conocer, para entender la forma en que funciona ROS, son los siguientes:

- ROS Máster (Nodo): este nodo proporciona el registro de nombre y consulta al resto del grafo de computación. Sin el Máster, los nodos no serían capaces de comunicarse entre ellos.
- Nodos: estos son procesos que ejecutan un código de programación específico y asignado en cada nodo, puede ser en distintos lenguajes de programación, entre los cuales se encuentran C++, Python, LISP, entre otros.
- Mensajes: para que exista comunicación entre nodos, es necesario el flujo de información, por este motivo, se crean los mensajes. Un mensaje es una estructura de datos compuesta por una combinación de tipos primitivos o mensajes. Los tipos primitivos de datos como, por ejemplo, entero, punto flotante, booleano y otros están soportados, además de extensiones de aglomeraciones de ellos, conocidos como arreglos. Los mensajes pueden incluir estructuras y arreglos.
- Tópicos: son los canales a través de los cuales se comunican los nodos, mediante un sistema de transporte basado en la semántica publicar/subscribir. Un nodo envía mensajes publicando a un tópico. El tópico es un nombre que se usa para identificar la ruta del mensaje. Un nodo que está interesado en cierto tipo de datos se subscribirá a un tópico específico. Pueden existir muchos publicadores simultáneamente para un solo tópico y un nodo puede publicar o subscribirse a múltiples tópicos.
- Servicios: el modelo de publicar/subscribir es un paradigma de comunicación útil y flexible, pero, en muchos casos, el transporte en un único sentido no es suficiente para las interacciones de petición y respuesta que a menudo se requieren en un sistema distribuido. La petición y respuesta se realiza a través de los servicios, que se definen a partir de una estructura de mensajes: una para la consulta y otra para la respuesta. Un nodo proporciona un servicio con un nombre y un cliente utiliza dicho servicio mediante el envío del mensaje de petición y espera por la respuesta. Este sistema es útil en casos cuando es necesario ejecutar un módulo que tarda bastante

tiempo en inicializarse; de esta manera, solo es necesario levantar el sistema como servicio una vez y que esté esperando por una consulta por parte de un cliente y evitar gastar tiempo valioso.

2.11 Cámara 3D

En el mundo de la robótica, es de gran importancia obtener información sobre lo que rodea a los robots físicamente, con el objetivo de conocer su posición en el espacio, además de tomar decisiones con base en esta información, por ejemplo, conocer la posición de un objeto, evitar obstáculos, entre otros. En este proyecto, dicha información se adquiere de un cámara 3D, modelo ASUS XTION, la cual detecta objetos en un rango entre 0.8 metros y 3.5 metros, compuesto por un emisor y un receptor.

La forma en que funcionan estas cámaras 3D se debe a que el emisor de luz infrarroja es capaz de generar un patrón de puntos, como se observa en la figura 2.35, organizados en una matriz similar a la que se muestra en la figura 2.36, donde cada casilla de esta matriz corresponde a una aglomeración de puntos en una configuración única y diferente entre todas. De esta manera, cuando el receptor detecta alguna de estas aglomeraciones de puntos, basados en su tamaño y orientación, es capaz de calcular la distancia a la cual se encuentra esa casilla de la matriz.

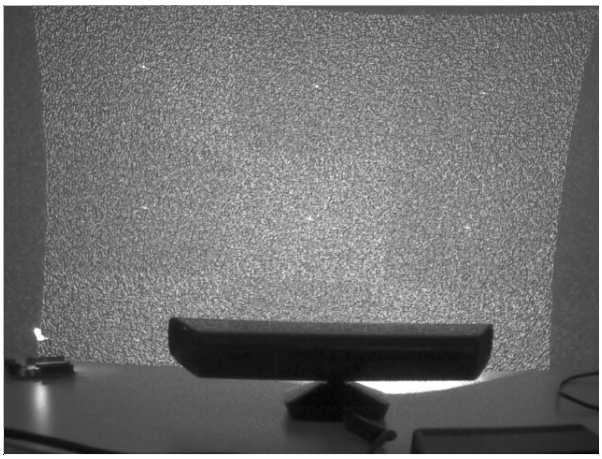


Figura 2.35: Patrón de puntos generados por la cámara 3D [28].

En el ejemplo de la figura 2.37, se muestra el patrón de puntos generado por la cámara XTION, la flecha roja se refiere a la casilla número 130 de la matriz y la flecha azul se refiere a que el sensor ha detectado esa casilla a una distancia dada. De esta manera, estas cámaras 3D son capaces de brindarle información a los robots de lo que se encuentra en frente de ellos. Es importante advertir que estas cámaras funcionan bien sin contaminación de luz infrarroja, en otras palabras, se debe evitar utilizarlo en exteriores debido al sol y evitar utilizarlas cerca de otras cámaras que utilicen esta misma tecnología.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139
140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179
180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199
200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219
220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259
260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279
280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299
300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319
320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339
340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359
360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379
380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399

Figura 2.36: Patrón de puntos representados por una matriz de números.

2.12 Reconocimiento de voz

El reconocimiento de voz es una rama interdisciplinaria de la lingüística computacional, la cual desarrolla metodologías y tecnologías que permiten transformar una señal acústica a una cadena de palabras mediante una computadora [29]. A este campo de investigación también se le conoce como *reconocimiento automático de voz, reconocimiento de voz por computadora* o simplemente *voz a texto*, la cual incorpora conocimiento e investigación en los campos de la lingüística, la informática y la ingeniería eléctrica.

El reconocimiento de voz es una de las herramientas que aporta más peso para este proyecto en el tema de asistencia a pacientes, ya que el paciente no necesita aprender a utilizar algún dispositivo tipo tableta o similar, sino que el robot se adapta a los pacientes mediante la comunicación por voz. Estos sistemas consisten, básicamente, en un micrófono que es utilizado por un computador para grabar un audio, en este caso, voz de una persona; luego, este audio se procesa en un sistema de reconocimiento de voz y se obtiene como resultado la mejor aproximación del audio traducido en texto. Posteriormente, se explica cuáles herramientas computacionales se utilizan para hacer frente a este problema.

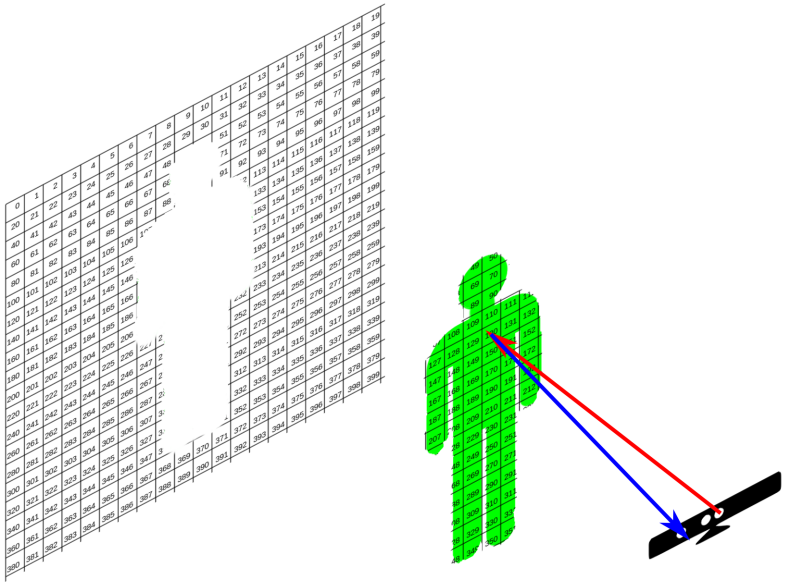


Figura 2.37: Representación de cómo funciona el cálculo de profundidad con cámara 3D XTION.

Capítulo 3

Integración de herramientas

Este proyecto integró, mediante *software* tipo *middleware*, distintas disciplinas como visión artificial, aprendizaje profundo, mapas de profundidad, entre otros. Para dar una noción de cómo se solucionó el problema, a continuación, se presenta una breve descripción del algoritmo implementado. Posteriormente, se ampliará con más detalle.

La primera tarea fue reconocer los objetos específicos en el entorno que rodea al robot. Para realizarlo, se utilizó el *framework* de *deep learning* llamado Caffe, el cual soporta diferentes tipos de arquitecturas de redes neuronales artificiales, en este caso, se utilizó el proyecto *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* [30], el cual está orientado a la clasificación de objetos en imágenes y la ubicación de la región donde este se encuentre. Básicamente, Caffe se encarga de recibir como *entrada* una imagen tomada por el robot y su *salida* son los objetos reconocidos por la red neuronal artificial. Es importante destacar que los objetos que tuvo que reconocer el robot no fueron definidos desde el inicio del proyecto, debido a la incertidumbre de cuáles podían tomar los brazos del robot Pepper, debido a las limitaciones de movimiento de cada articulación presente en él.

La segunda tarea fue implementar el reconocimiento de la voz del paciente y traducirlo como una instrucción de texto. Para lograrlo, se utilizó la biblioteca SpeechRecognition en Python y las aplicaciones Google Speech Recognition y Google Cloud Speech, dedicadas a esta función. Esta biblioteca tuvo la función de convertir el mensaje de voz del paciente a texto y, con base en este, enviarle una instrucción al robot. Se debe aclarar que, después de la conversión de audio a texto, el algoritmo atendió a instrucciones y objetos limitados, con el objetivo de simplificar el proyecto, debido al corto período para desarrollarlo. Las palabras que se tuvieron que reconocer, según las métricas definidas, son los verbos traer y alcanzar, junto con los sustantivos de los objetos por reconocer.

La tercera tarea fue integrar el desplazamiento y movimiento del robot, la sujeción de los objetos, el reconocimiento de objetos y de voz. Lo primero que se hizo fue colocar un micrófono cerca del paciente, el cual cumplió la función de siempre estar escuchando lo que decía y reconocer las instrucciones de voz, ya que, si se utilizaban los micrófonos

del robot, era posible que la voz del paciente no tuviera la intensidad suficiente para activarlos.

Luego de que el sistema tenía definido el objeto que debía buscar en el entorno, la siguiente etapa fue realizar un muestreo de todo el entorno alrededor del robot, haciendo uso del reconocimiento de imágenes. Si no se reconocía ningún objeto, el robot no debía hacer nada; en cambio, si se reconocía algún objeto, el algoritmo continuaba a la etapa de acercarse al objeto, donde se utilizó una cámara 3D, para obtener información del entorno que rodeaba al robot y luego procesarla mediante *software*. Luego de que se localizaba el objeto, el robot debía desplazarse hasta las cercanías de este, tomarlo mediante un movimiento pregrabado (con el fin de evitar el tema de *grasping*), desplazarse hasta donde se encontrara el paciente y colocarlo cerca de la persona.

Las restricciones que se asumieron para llevar a cabo este proyecto contemplan que la altura a la que se debía encontrar el objeto está en un rango entre 70 cm a 100 cm medidos desde el suelo de la habitación. Los movimientos necesarios para tomar el objeto fueron pregrabados. El proyecto no tenía como requisito contemplar obstáculos, tampoco que su evaluación fuera en un entorno real como un cuarto de habitación, un cuarto de hospital, entre otros; esto por el corto período para desarrollar el proyecto. Por esta razón, la evaluación del proyecto se llevó a cabo en el laboratorio RoViT, en la Universidad de Alicante, España.

Al estar compuesto por diferentes herramientas que el investigador desconocía, a excepción de un par de lenguajes de programación (C++ y Python), fundamentos de robótica, física y matemática, el primer paso para realizar este proyecto fue comprender en qué consistía cada una de estas herramientas, con el fin de tener una noción de cómo serían integradas para que interactuaran de forma sincronizada y correcta. Una observación importante sobre la forma de describir la solución es que los pasos que se muestran en las siguientes secciones describen las etapas que fueron necesarias para llegar a la solución final, empezando desde lo más básico, como entender la forma en que funcionaba cada herramienta de forma individual, hasta llegar a la integración de herramientas que hacen de este proyecto un sistema funcional.

3.1 Clasificación de objetos

La primera herramienta que se estudió se enfoca en la detección de objetos basada en el proyecto Faster R-CNN. La implementación de este artículo se obtuvo del código de programación llamado py-faster-rcnn [31], el cual constituye un *script* de Python que invoca funciones de Caffe. Debido a que todas las herramientas de este proyecto fueron comunicadas entre ellas mediante ROS, se tuvo que modificar este para que funcionara en el *software* de tipo *middleware*. Este procedimiento fue sencillo, debido a la compatibilidad que posee ROS para ejecutar distintos lenguajes de programación, el cual, en este caso, es capaz de utilizar todos los paquetes de Python.

Para una mejor comprensión, se va a partir del hecho de que el recurso computacional de *hardware* era suficientemente potente para procesar en tiempo real todas las imágenes. Si este hubiera sido el caso, como se muestra en la figura 3.1, el nodo llamado RCNN Tagger se suscribía al tópico /camera para obtener las imágenes de la cámara web de la portátil. Una vez que se recibía una imagen, este nodo RCNN Tagger hacía un llamado como cliente al servicio Classifier Server, de manera sincronizada, donde la consulta del servicio consistía en la imagen y la respuesta del servicio consistía en un mensaje del tipo PredictionsList, el cual contenía información de los objetos reconocidos.

Una vez que el servicio respondía, el nodo RCNN Tagger se encargaba de interpretar este mensaje y publicaba al tópico /publishinfo, de manera que, si no se detectaba algún objeto, nada más se mostraba la imagen en la pantalla; si se detectaba algún objeto, se mostraba en pantalla la imagen con los respectivos *bounding box* dibujados de cada objeto y, en la terminal, se mostraba el *label* (etiqueta), *score* (puntuación) y las coordenadas en píxeles de las dos esquinas del rectángulo que encierra al objeto reconocido.

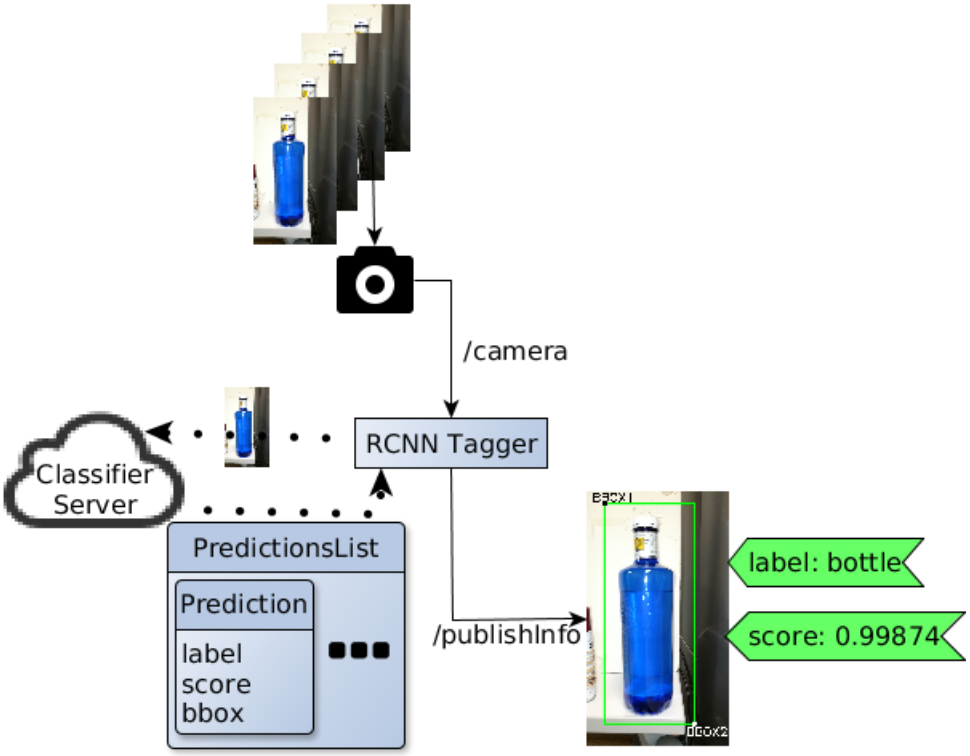


Figura 3.1: Caso ideal del reconocimiento de objetos mediante *Faster R-CNN*.

Ahora bien, debido a que, para un CPU Intel Core i5 sétima generación, el proceso de realizar inferencias sobre una imagen e identificar objetos con el nodo Classifier Server tomaba alrededor de 18 segundos, se tuvo que modificar la arquitectura anterior, tal como se muestra en la figura 3.2. Al cambiar el enfoque a este caso real, este sistema consistió en que la cámara web de la portátil tomaba imágenes del entorno a 30 cuadros por segundo. Debido a que el servicio de clasificación de objetos no procesaba las imágenes en tiempo real, se colocó el nodo Throttler (*throttle* significa "regular el flujo" en inglés), el cual se

encargó de regular la cantidad de imágenes que se enviaban al nodo RCNN Tagger, donde se podía regular que este nodo recibiera una imagen cada 20 segundos, lo cual se traduce como 0.05 cuadros por segundo.

De esta manera, el cliente RCNN Tagger era capaz de realizar consultas al servicio de clasificación de objetos, Classifier Server, sin saturarlo de consultas. Una vez que se realizaban las inferencias sobre la imagen, se obtenía el mismo tipo de respuesta que en el caso anterior, con su *label*, *score* y las coordenadas en píxeles de las dos esquinas del rectángulo que encierra al objeto reconocido.

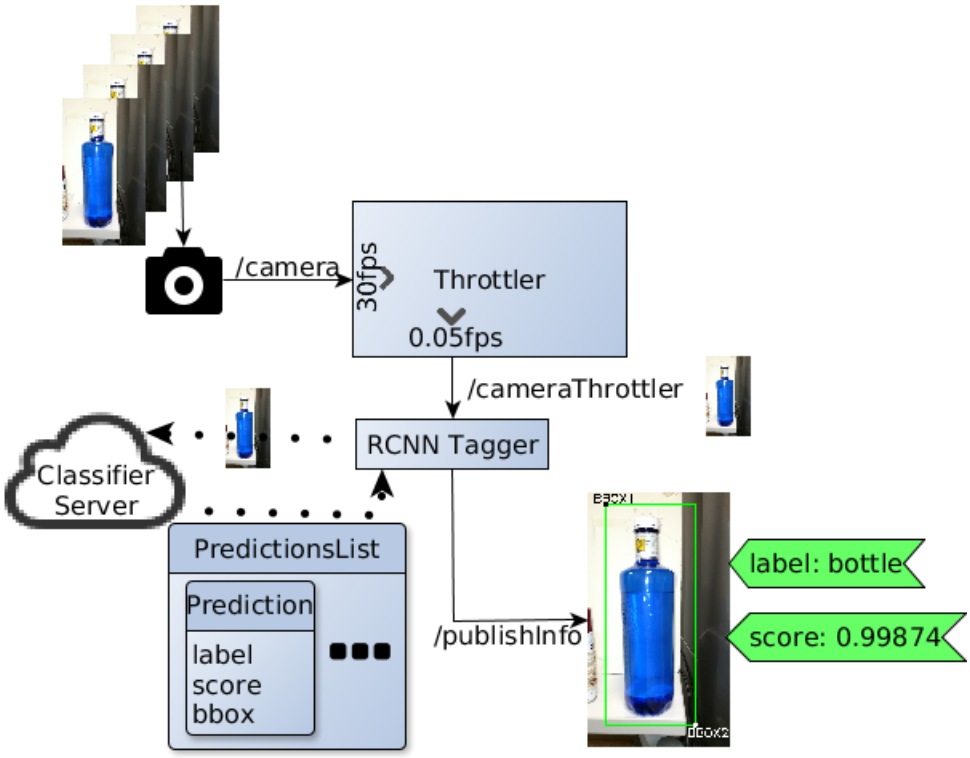


Figura 3.2: Método de detección de objetos utilizando ROS.
Caso real.

Cuando se analizó esta propuesta, un inconveniente fue la velocidad con la que se realizaba el reconocimiento de los objetos, ya que, para realizar una búsqueda de objetos en todo el entorno del robot, cabe la posibilidad de que fueran 20 imágenes en total (considerando un barrido 360°), lo cual se traduce en que un solo mapeo tardaría 400 segundos. Esto fue considerado un proceso lento para solo buscar el objeto, aunque no había ningún tiempo límite para completar el objetivo final.

Debido a que este tiempo se quiso mejorar, para lograr un mejor resultado en cuestión de tiempo, la solución a este problema fue utilizar una GP-GPU *General-Purpose Computing on Graphics Processing Unit*, la cual logró realizar inferencias a una velocidad de 115 milisegundos por imagen en promedio. Esto significa que pudo procesar 156 veces más rápido que un CPU del modelo mencionado anteriormente. Más adelante, se presenta un detalle más amplio de la integración de una GP-GPU para disminuir el tiempo de

inferencia.

3.2 Empleo de cámaras 3D

La siguiente tarea fue comprender cómo funcionaban las cámaras 3D y aprender a manipular la información que suministraban. Para esta etapa, se utilizó una cámara 3D modelo Kinect versión 1.0 de Microsoft. Se debe suponer que lo que existía en frente de la cámara 3D es lo que se muestra en la figura 3.3a, representado por una imagen de tres canales del tipo RGB. La primera representación que se obtuvo fueron las imágenes o mapas de profundidad, tal como se muestra en la figura 3.3b, las cuales son imágenes en escala de grises y de un solo canal, donde cada píxel en esa matriz de la imagen está asociado con su profundidad en el espacio, es decir, los mapas de profundidad se construyen basados en las profundidades de cada píxel en el campo de visión del robot.

Si no se pudo obtener información en un píxel específico, este se representa por un color totalmente negro y en caso de que sí se pudo determinar la distancia en un píxel específico, cuanto más cerca se encuentre el objeto, el píxel tendrá una mayor intensidad de color negro y, por el contrario, a mayor distancia, el píxel tendrá una menor intensidad de color negro. Esto se puede observar en la figura 3.4, donde los elementos NaN (en inglés *Not a Number*) son información que no pudo ser determinada y los demás números son las profundidades de cada píxel en metros; esta es la forma como se interpreta un mapa de profundidad a nivel de manipulación de imágenes digitales.

(a) Imagen RGB. (b) Imagen mapa de profundidad.

Figura 3.3: Entorno presente frente a Kinect v1.0.

La otra representación del entorno que se obtuvo de las cámaras 3D son las nubes de puntos, definidas como un conjunto de puntos en un sistema de coordenadas tridimensional, dichos puntos son definidos, generalmente, mediante coordenadas X, Y y Z, además, están destinados a representar la superficie externa de un objeto. Mediante un visualizador incorporado en ROS llamado RViZ, el entorno de la figura 3.3a se puede observar en su representación de nube de puntos mostrada en la figura 3.5.

El objetivo de relacionarse con las cámaras 3D fue establecer cuál de las dos represen-

1.12	1.10	1.15	1.12	NaN
NaN	1.12	1.11	1.12	1.12
1.12	1.12	1.10	1.12	1.12
1.14	1.12	1.12	1.08	NaN
1.05	1.12	NaN	1.19	1.12

Figura 3.4: Representación de mapa de profundidad.

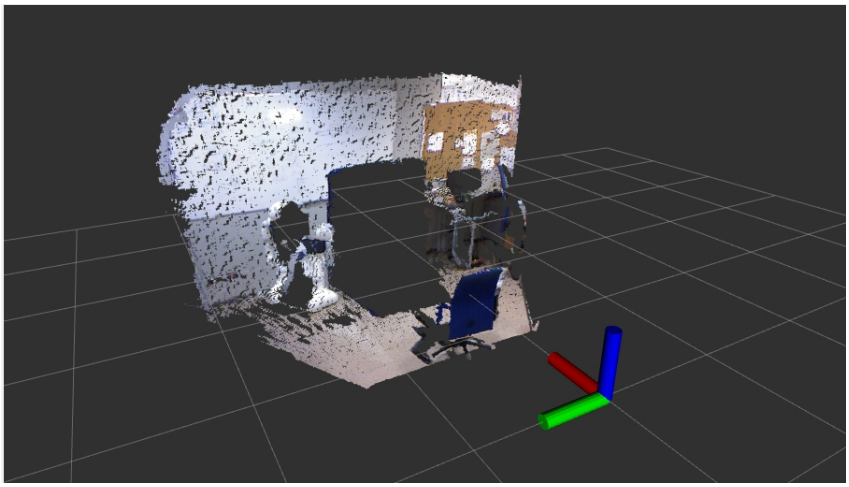


Figura 3.5: Nube de puntos del entorno mostrado en la figura 3.3a.

taciones del entorno funcionaba de mejor forma, para determinar la distancia a la cual se encontraba cada objeto en el campo de visión del robot. De esta forma, junto con la herramienta anterior Py-Faster-RCNN, se pudo localizar el objeto en el espacio. La ventaja observada sobre las nubes de puntos es que pueden ser procesadas en un sistema coordenado de tres dimensiones y permiten segmentar los objetos en el entorno mediante *software*, por ejemplo, PCL (*Point Cloud Library*).

Cuando se analizaron las dos opciones, se observó que los mapas de profundidad poseen la característica de que se encuentran alineados con las imágenes RGB. Debido a este alineamiento de imágenes y recordando que el servicio de clasificación de objetos utiliza la imagen RGB, se decidió que era más sencillo trabajar con los mapas de profundidad, porque, si se recuerda, la respuesta del servicio de clasificación brinda el *bounding box*, que en palabras simples es un rectángulo que encierra al objeto reconocido. Este se puede reflejar en el mapa de profundidad y realizar un recorte con este mismo, con la intención de promediar la profundidad de todos los píxeles dentro de este rectángulo y lograr un estimado de la profundidad del objeto.

Utilizando la lógica mencionada en el párrafo anterior, se desarrolló un sistema que calculaba la distancia aproximada a la que se encontraba un objeto, de manera preliminar, definiendo el *bounding box* de forma manual. Es importante aclarar que un mapa de profundidad brinda la distancia de profundidad de cada píxel, pero no es capaz de segmentar cada objeto, por ejemplo, decir que la pared se encuentra a 3 metros y una silla está a 2.5 metros. Debido a que el robot debía tener una noción de dónde estaba localizado el objeto en el espacio, se ideó una forma de integrar la detección de objetos y estos mapas de profundidad. Esto se logró al capturar una imagen RGB y el mapa de profundidad, ambas adquiridas desde el Kinect en el mismo instante, por ejemplo, una imagen RGB mostrada en la figura 3.6a y un mapa de profundidad mostrado en la figura 3.6b, respectivamente.

(a) Imagen RGB.

(b) Imagen mapa de profundidad.

Figura 3.6: Imágenes del objeto en frente de la cámara 3D para el cálculo de profundidad promedio.

El siguiente paso fue simular de modo manual el *bounding box* de un objeto, como si hubiera sido reconocido por el servicio de clasificación de objetos, tal como se muestra en la figura 3.7. La razón por la que se decidió realizar este proceso de forma manual fue para agilizar el proceso de promediar la distancia a la cual se encontraba el objeto, ya que, si se intentaba incorporar el servicio de clasificación, se hubiera requerido que primero se reconociera el objeto con un tiempo de ejecución de 20 segundos y luego que calculara la profundidad con un tiempo de ejecución de 100 milisegundos, lo cual representaba una pérdida de tiempo innecesario.

Dentro de los mapas de profundidad, es importante indicar que todos los elementos del tipo NaN (*Not a Number*) fueron descartados, para que se pueda efectuar la media aritmética de las distancias. Con el fin de que este procedimiento quede más claro, se debe suponer que la imagen de la figura 3.8 está representada en la matriz de la figura 3.9. Las distancias de cada píxel del objeto reconocido se encuentran dentro del *bounding box* marcado por un rectángulo rojo. Descartando los NaN, la distancia aproximada del objeto corresponde a 0.8082 metros, aplicando una media aritmética.

El siguiente paso fue comunicar los servicios de reconocimiento de objetos y cálculo de promedio de profundidad de un objeto, para que calculara la distancia del objeto deseado.

Figura 3.7: *Bounding box* del objeto de interés para el cálculo de profundidad.

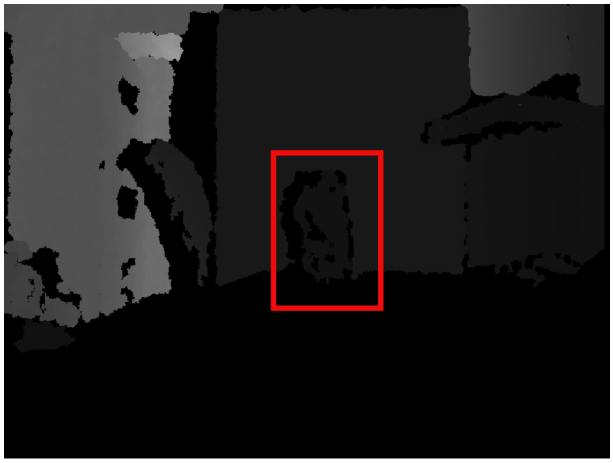


Figura 3.8: *Bounding box* que rodea el objeto deseado en el mapa de profundidad.

Para tener una idea más clara de cómo funcionó, se debe suponer el siguiente entorno mostrado en la figura 3.10.

Para llevar a cabo el proceso de reconocer el objeto y obtener su distancia promedio de forma automática, se implementó la lógica mostrada en la figura 3.11. Lo primero que se realiza en este diagrama es que el nodo Throttler se subscriba a los tópicos de imagen RGB e imagen de mapa de profundidad, cada 20 segundos y las envíe al nodo RCNN Tagger. Una vez que este paquete de imágenes se encuentra en el RCNN Tagger, se realiza una consulta con la imagen RGB al servicio de clasificación de objetos, *Classifier Server*, donde el modelo de la red es entrenado con el conjunto de datos Pascal VOC [22] y se obtiene como respuesta una lista de predicciones, ya sea con contenido relevante de los objetos o vacío.

Si no se detectó ningún objeto, no se realiza ninguna consulta al servicio Profundidad

0.84	0.78	0.83	0.8	0.83	0.83	0.83	0.8	0.83	0.84	0.84	0.83	0.8	0.83	0.82	0.83	0.83	0.83	0.81	0.84
0.78	0.79	0.8	0.83	0.83	0.81	0.84	0.78	0.82	0.84	NaN	0.78	0.83	0.81	0.81	0.79	0.79	0.81	0.79	0.84
0.79	0.81	0.84	0.83	0.83	0.84	0.79	0.8	0.8	0.82	0.83	0.81	0.81	0.8	0.83	0.79	NaN	0.8	0.81	0.82
0.81	0.8	0.78	0.82	0.81	0.83	0.78	0.82	0.82	0.81	0.78	0.79	0.84	0.82	0.81	0.78	0.8	0.79	0.79	0.79
0.8	0.8	NaN	0.8	0.79	0.79	0.83	0.79	0.83	0.79	0.82	0.79	NaN	0.82	0.78	0.79	0.79	0.79	0.79	0.81
0.81	0.84	0.79	0.83	0.84	0.78	0.83	0.78	0.83	0.82	0.84	0.84	0.8	0.8	0.83	0.78	0.83	0.81	0.82	0.83
0.8	0.84	0.79	0.78	0.84	0.83	0.83	0.82	0.82	NaN	0.79	0.79	0.78	0.79	0.84	0.83	0.8	0.78	0.82	0.84
0.82	0.82	0.8	0.78	0.82	0.78	0.81	0.84	0.78	0.79	0.82	0.78	0.79	0.81	0.84	0.81	0.82	0.84	0.79	0.79
0.8	0.79	0.81	0.8	0.8	0.84	0.82	0.79	0.79	0.81	0.78	0.83	0.78	0.83	0.81	NaN	0.83	0.82	0.78	0.84
0.84	0.84	0.83	0.84	0.84	0.83	0.84	0.8	0.78	0.78	0.81	0.79	0.82	0.82	0.78	0.78	0.82	0.8	0.79	0.83
0.79	0.83	0.78	0.82	0.8	NaN	0.83	0.83	0.83	0.82	0.84	0.83	0.84	0.8	0.82	0.84	0.79	0.84	0.81	0.82
0.8	0.8	0.79	0.78	0.8	0.79	0.79	0.78	0.8	0.78	0.8	0.84	0.8	NaN	0.78	0.84	0.83	0.78	0.78	0.84
0.79	0.82	0.78	0.8	0.78	0.84	0.81	0.79	0.8	0.8	0.82	0.78	0.78	0.81	0.79	0.84	0.78	0.79	0.82	0.8

Figura 3.9: Representación numérica de distancias de la figura 3.8.

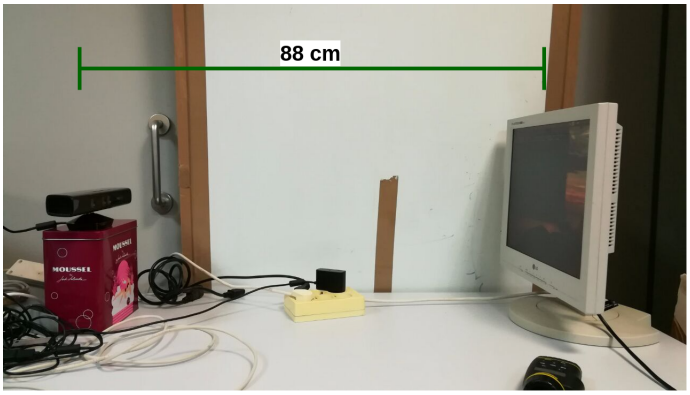


Figura 3.10: Entorno de prueba para la integración del reconocimiento de objetos y cálculo de su profundidad promedio.

Server. Por otro lado, si se detectó un objeto, se debe realizar una consulta a este servicio Profundidad Server encargado de promediar las distancias de cada píxel dentro del *bounding box*, donde la solicitud está compuesta por el *bounding box* de los objetos reconocidos y el mapa de profundidad contenido en el paquete de imágenes respectivo. Una vez que este servicio ha procesado la información, este responde con el *label, score, bounding box* y la distancia promedio a la cual se encontraba ese objeto, tal como se muestra en el nodo Resultado del diagrama en cuestión.

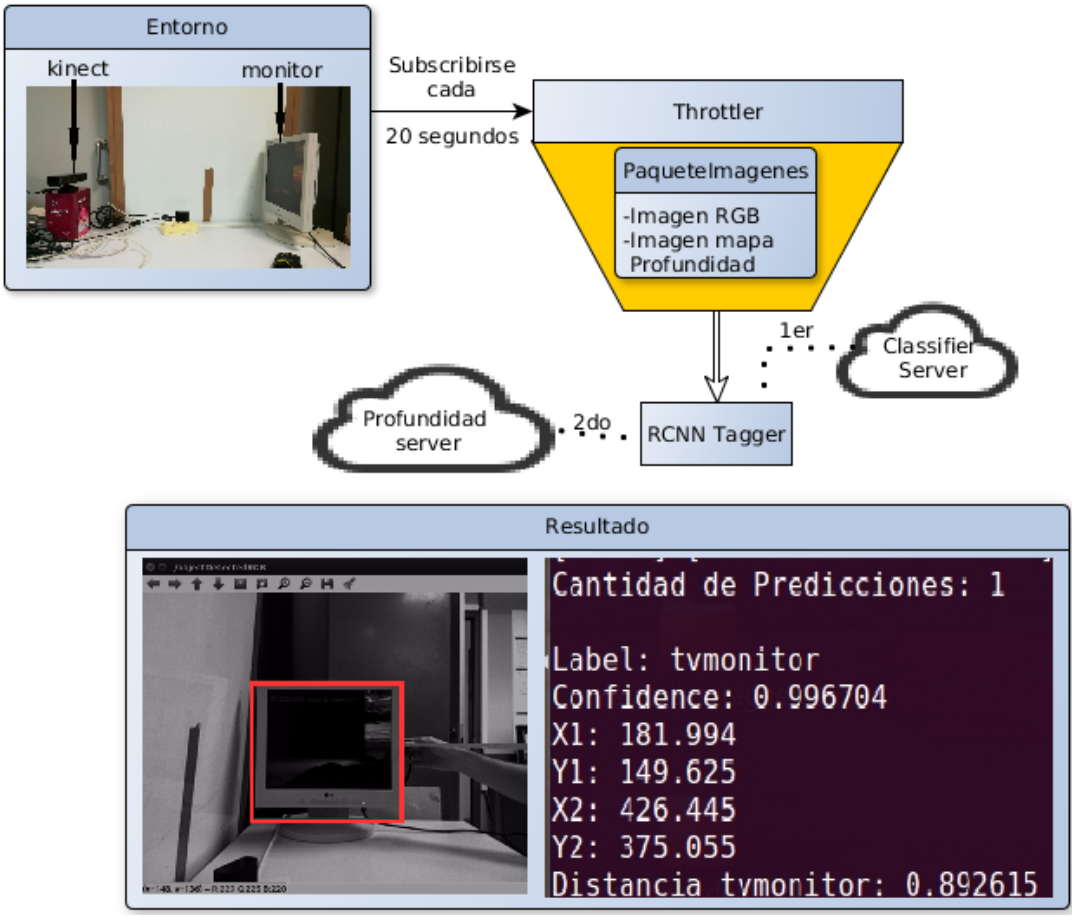


Figura 3.11: Integración de los servicios de detección de objetos y cálculo de profundidad.

3.3 Reconocimiento de voz

Otra herramienta que se utilizó fue el reconocimiento de voz, cuyo objetivo fue reconocer la voz del paciente que se encontraba situado en un lugar fijo. Este módulo se basó en una biblioteca para Python llamada Speech Recognition v3.8.1 [32], que, además, puede ser utilizada en ROS. Para esta solución, se utilizaron dos aplicaciones de Google, la primera fue Google Speech Recognition que reconocía el idioma inglés y es gratuita; la otra se llama Google Cloud Speech API que reconocía el idioma español, esta es de pago, con un precio de $0.006 por cada 15 segundos de audio de consulta.

Debido a que el sistema siempre debía estar escuchando el entorno que lo rodeaba, si solo se utilizaba Google Cloud Speech API (de pago), esto hubiera provocado que el costo aumentara, debido a que hubiera sido necesario procesar todo lo que escuchaba. Como consecuencia, para solucionar este problema, se utilizó Google Speech Recognition que es gratuito, el cual se encargó de esperar por un *trigger* o disparo en idioma inglés, en este caso, el sistema espera escuchar "okay Pepper", para que, después de que se escuchara esta señal, entrara a funcionar Google Cloud Speech API e interpretara la instrucción en español, por ejemplo, "traer botella". En la figura 3.12 se muestra un diagrama de cómo

funciona este sistema.

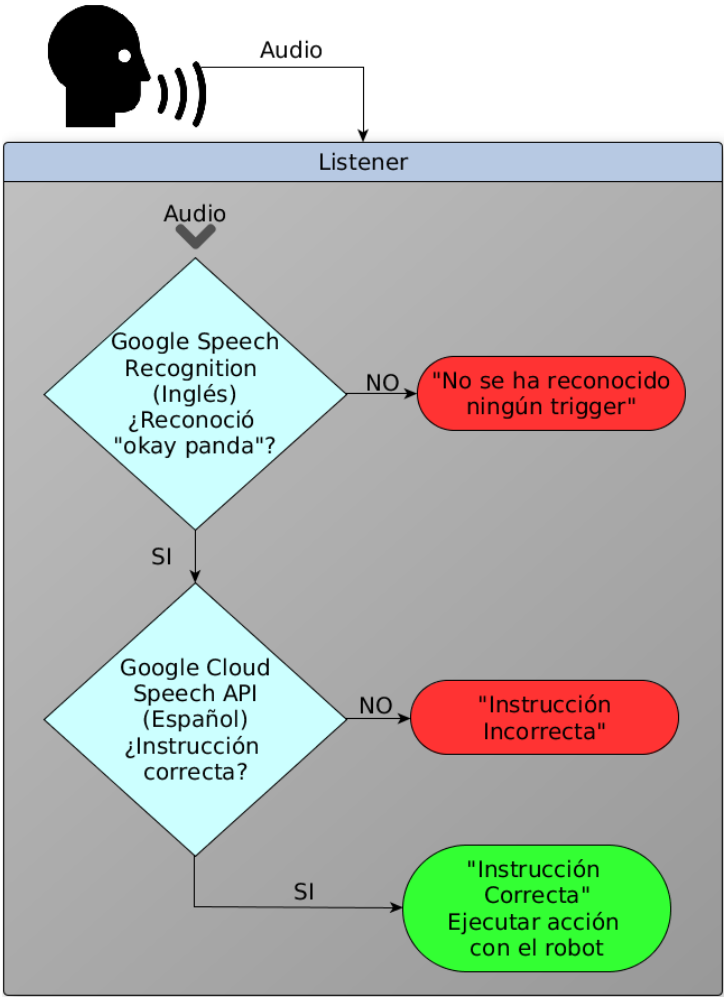


Figura 3.12: Diagrama de flujo implementado para la etapa de reconocimiento de instrucciones de voz.

La forma en que esta biblioteca de reconocimiento de voz segmenta cada instrucción se basa en un límite o *threshold* del nivel de energía de sonido del entorno, por lo tanto, cuando se sobrepasa este límite, la grabación del audio del entorno inicia y cuando la intensidad del audio cae debajo de este *threshold*, la grabación se detiene. Luego, este audio es lo que se envía como consulta a los servicios Google Speech Recognition y Google Cloud Speech API. Este comportamiento se observa en la figura 3.13, donde la zona coloreada en azul que se localiza sobre el *threshold* es el audio importante.

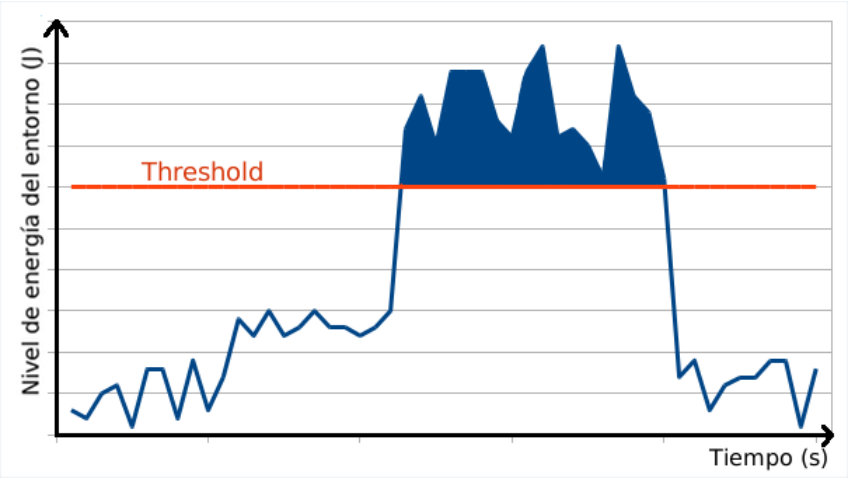


Figura 3.13: Grabación de audio cuando se supera el *threshold*.

3.4 Entrenamiento de una red para la detección de una botella

En esta etapa del proyecto, se decidió elegir objetos que cumplieran con las condiciones de tamaño, para que los brazos y manos del robot lograran sujetarlos de manera segura. Por lo tanto, la siguiente tarea fue entrenar una red que al menos reconociera uno de esos objetos candidatos, para que las pruebas se dirigieran hacia el objetivo final. Por esta razón, se decidió crear un conjunto de datos basado en una botella azul, como la que se muestra en la figura 3.14, para entrenar una red que fuera capaz de reconocerla.

La razón por la que se eligió esta botella es porque se demostró, mediante una prueba rápida, que el robot era capaz de sujetarla mediante una secuencia de movimientos definidos manualmente. Esta prueba se realizó mediante un software intuitivo llamado Choregraphe, el cual lo suministra el fabricante del robot.

Figura 3.14: Botella para preparar el conjunto de datos.

El procedimiento que requirió la creación de este conjunto de datos inició por definir los objetos que se deseaba reconocer, para este primer caso, la botella azul. Lo siguiente fue tomar alrededor de 1000 imágenes por cada objeto en diferentes entornos y puntos de vista.

Para agilizar este procedimiento, se tomaron vídeos con una cámara con estabilizador y luego, mediante el *software* ffmpeg, fueron extraídas todas las imágenes del video.

Cuando se obtuvo el conjunto de datos, continuó la etapa de etiquetado de las imágenes. Este procedimiento consistió en crear un archivo donde se le muestra al algoritmo de aprendizaje información relevante sobre el *ground truth* de cada imagen, este término se refiere a información proporcionada por observación directa, en otras palabras, es indicarle a la red, mediante un rectángulo, dónde se encuentra el objeto, para que aprenda a reconocerlo. Para suministrar esta información a la red, se creó un archivo formato *.txt*, donde se le indicó en orden, primero la ruta de almacenamiento de cada imagen en el equipo, segundo la categoría del objeto y tercero las coordenadas de las esquinas superior izquierda e inferior derecha del rectángulo que encierra al objeto. Para agilizar este procedimiento y evitar errores, se utilizó un *script* de Python. Este proceso de generación del conjunto de datos y etiquetado tomó alrededor de 12 horas para estas 1000 imágenes.

Cuando ya se había etiquetado el conjunto de datos, continuó la etapa de entrenamiento de la red. Es importante tomar en cuenta que, del conjunto de datos generado, alrededor del 90 % fue utilizado para la etapa de entrenamiento y el otro 10 % se utilizó para la etapa de evaluación de la red. Este entrenamiento se realizó utilizando el modelo *multi-class* de Faster R-CNN en Caffe, durante 40 000 iteraciones, utilizando los pasos mencionados en la sección 2.8. Se utilizó el *solver* denominado *Descenso por Gradiente Estocástico con Nesterov*, con un *learning rate* de 0.001 y un *momentum* de 0.9.

3.5 Mejorar el tiempo de inferencias en la clasificación de objetos

Como se mencionó previamente, debido a que el tiempo para realizar inferencias con un CPU Intel Core i5 sétima generación tomaba aproximadamente 18 segundos por imagen, se buscó la forma de disminuir este lapso. Por esta razón, se utilizó un computador llamado en el laboratorio como JACKSON, que posee una poderosa GP-GPU modelo NVIDIA GeForce GTX 1080 Ti, la cual reduce este tiempo a 115 milisegundos. Para comprender cómo se implementó esta adaptación, se debe observar la figura 3.15.

Esta configuración consiste en que, cuando la imagen llega al nodo RCNN Tagger, esta es enviada al nodo Classifier Server, pero, a diferencia de la figura 3.2 donde la inferencia se realizaba en el CPU de la máquina local, este nodo hace una consulta a la dirección 172.18.33.118:5000/index administrada por JACKSON, utilizando Flask (*framework* para desarrollar aplicaciones web desde Python), en el cual la consulta es la imagen donde se desea buscar un objeto y la respuesta está compuesta por la etiqueta, la probabilidad y el *bounding box* de los objetos reconocidos. De esta manera, cuando JACKSON publica a la dirección antes mencionada, el nodo Classifier Server escucha esta respuesta y se la transmite al nodo RCNN Tagger. Es claro que el objetivo fue el mismo, reconocer objetos dentro de una imagen, lo que cambió fue la velocidad con la que se llevaron a cabo las

inferencias.

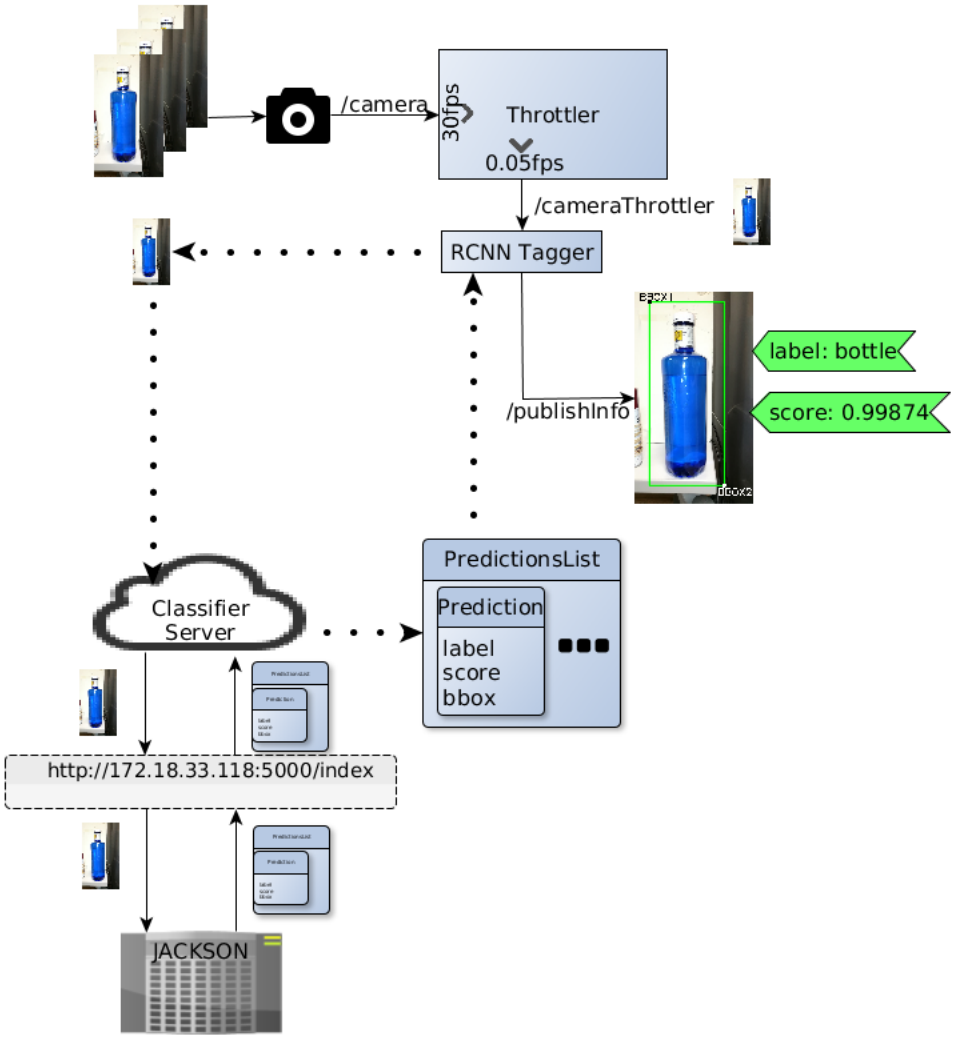


Figura 3.15: Integración del computador JACKSON que reduce el tiempo que toman las inferencias.

3.6 Conexión entre ROS y el robot Pepper (NAOqi)

Otra tarea pendiente fue comunicar ROS con el robot Pepper, lo cual se logró aprovechando el mayor fuerte de ROS, la abstracción de *hardware*. Para lograr esto, primero se utilizó un puente entre el sistema operativo interno del robot llamado NAOqi y ROS, llamado NAOqi-Bridge [33], el cual permitió obtener información relevante del robot, pero también leer información de los sensores y manipular sus actores.

Cuando se logró la comunicación de ROS con el robot, se decidió que la primera tarea sería mover los brazos del robot. Para lograr esto, fue necesario publicar a un nodo encargado de manipular las articulaciones del robot, el nombre de las articulaciones y el ángulo al que se deseaban mover, respectivamente. Como ya se mencionó, todas las articulaciones

del robot tienen restricciones de movimiento que se tomaron en cuenta; a pesar de esto, si se publicaba un valor fuera de los rangos del movimiento, el sistema operativo interno del robot (NAOqi) no permitía el movimiento por seguridad.

Este proyecto tenía como requisito la traslación del robot entre distintas ubicaciones, lo cual se logró satisfactoriamente, al definir el desplazamiento deseado en dirección de los ejes *Pitch* y *Roll* de la trama de referencia del robot. Además de la traslación de la base, también se logró la rotación de la base alrededor del eje Yaw. Debido a que las pruebas con los servicios de clasificación de objetos y cálculo de profundidad se ejecutaron con una cámara 3D modelo Kinect versión 1.0, el siguiente paso fue utilizar la cámara 3D que poseía el robot modelo ASUS XTION, con un rango de funcionamiento que va desde 0.8 metros a 3.5 metros.

3.7 Mapeo del entorno alrededor del robot Pepper

Una de las condiciones que debía cumplir este proyecto era que, si al robot se le solicitaba un objeto, este debía buscar en todo el entorno y no solo en su campo de visión inicial. Por este motivo, se estableció un método que cumplía dicha condición, el cual consistió en que, primero, desde la posición aleatoria del torso del robot, se realiza un barrido con la articulación HeadYaw desde -110° a +110° con un paso de 22°, definido de esta forma para tomar imágenes con distintos puntos de vista de los objetos. Por tanto, este barrido se encargó de tomar fotos en cada paso y detectar objetos.

Si en ese primer barrido con la cabeza del robot no se detectaba el objeto deseado, el robot debía girar su base +90° alrededor del eje *Yaw*, para volver a realizar el barrido con su cabeza. Este ciclo fue programado para que se repitiera cuatro veces hasta que completara el barrido en 360° mediante la rotación de su base.

Si no se detectaba ningún objeto durante ese barrido, entonces el robot no ejecutaba ninguna acción. En cambio, si se detectaba un objeto, se debía alinear la base, el torso y la cabeza del robot, de manera que apuntaran en dirección del objeto deseado, con el objetivo de centrarlo en el campo de visión del robot. Esto se logró mediante un cálculo basado en el diagrama mostrado en la figura 3.16. En otras palabras, cuando el robot detectaba el objeto, el eje Roll de la base y el torso debían apuntar hacia el objeto y el ángulo HeadYaw debía regresar a su posición original, donde el valor de su ángulo regresaba a 0°, mirando en la dirección del eje Roll del torso.

En este caso, los ejes en color rojo *reference roll* y *reference pitch* son los ejes de referencia que se predefinen cuando el robot inicia, es decir, no están orientados con ningún sistema de referencia, por ejemplo, los polos magnéticos del planeta. Los ángulos mostrados en dicha imagen se detallan en la siguiente lista.

- Base: este es el ángulo al que apunta inicialmente el eje *Roll* del torso del robot, alrededor del eje Reference yaw (no mostrado en el diagrama y apuntado hacia

afuera de la página).

- Head: este es el ángulo al que apunta inicialmente el eje *Roll* de la cabeza del robot alrededor del eje HeadYaw (no mostrado en el diagrama y apuntado hacia afuera de la página).

- Corr. Img. ($\angle$CI): este ángulo, medido desde el centro del campo de visión del robot, se refiere a una aproximación utilizando la cámara del robot, cuyo aporte es importante para corregir el ángulo al que se debe establecer el próximo ángulo final del torso Set Angle.

- Set Angle: este es el ángulo al que debe quedar alineado el torso del robot (tomando en cuenta que el HeadYaw se estable a 0°), para que el objeto quede centrado en el campo de visión del robot.

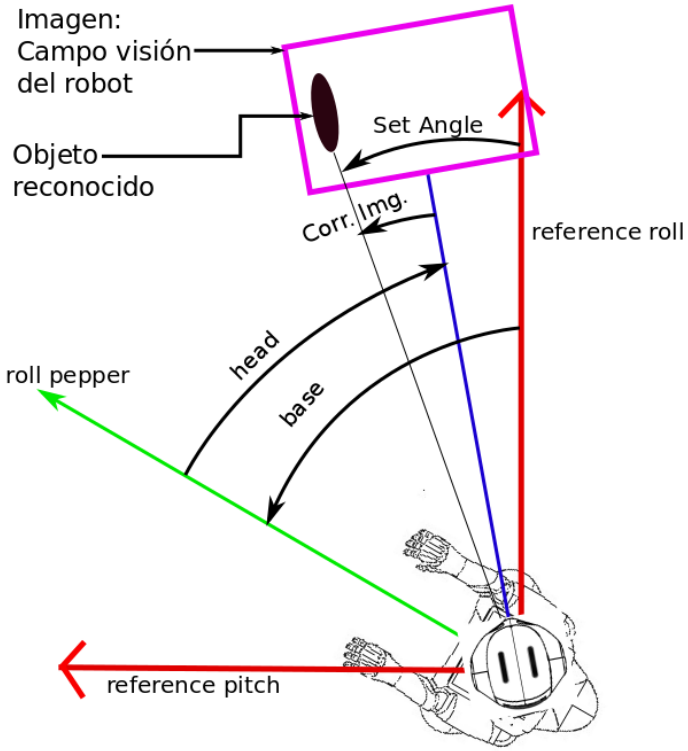


Figura 3.16: Diagrama para el cálculo de corrección de ángulo por imagen (Cuerpo del robot).

Los ángulos Base y Head se obtuvieron mediante NAOqi-Bridge, pero el ángulo Corr. Img debió ser calculado, donde se aplicó una relación lineal entre la longitud horizontal de la imagen tomada (620 píxeles) y el ángulo de apertura en el eje horizontal del campo de visión de la cámara (55.2°).

		Longitud horizontal de la imagen (px)	Ángulo campo de visión (°)
Ítem	1	640	-27.6
	2	0	27.6

Tabla 3.1: Coordenadas para cálculo de relación lineal entre la longitud horizontal de la imagen y el ángulo del campo de visión del robot.

El cálculo de la función lineal que pasa por esas dos coordenadas, con los datos de la tabla 3.1, se muestra en la ecuación 3.1, donde $\angle CV$ corresponde al ángulo de campo de visión del robot y I_a corresponde a dimensión horizontal de la imagen medido en píxeles.

$$\angle CV = 27.6 - \frac{55.2}{640} I_a \tag{3.1}$$

Para calcular esta corrección del ángulo por imagen, lo primero que se realizó fue calcular el centro del *bounding box* del objeto, reconocido en el eje horizontal de la imagen, tal como se muestra en la figura 3.17. De esta forma, en la ecuación 3.1 se renombraron sus variables para que cobrara sentido esta corrección de ángulo, mostrado en la ecuación 3.2, donde $\angle CI$ corresponde a la corrección de ángulo por imagen y C_{BBOX} corresponde al centro horizontal del *bounding box* medido en píxeles.

$$\angle CI = 27.6 - \frac{55.2}{640} C_{BBOX} \tag{3.2}$$

Una vez que se obtuvo esta corrección de ángulo por imágenes, fue posible calcular el ángulo donde se debía rotar el torso del robot, utilizando la ecuación 3.3, tomando en cuenta que todos los ángulos se sumaban, porque la regla de la mano derecha y la aproximación por corrección de imagen suministraban el signo al ángulo.

$$\angle Set\ Angle = \angle base + \angle head + \angle CI \tag{3.3}$$

3.8 Acercamiento del robot al objeto

El siguiente paso fue acercar al robot hasta el objeto a una distancia de un metro. Esto se logró con un procedimiento iterativo, mediante el uso de los servicios de detección de objetos y cálculo de profundidad, el cual consistió en, primero, ubicar el objeto en el campo de visión de la cámara del robot; segundo, centrar el objeto en el campo de visión del robot; tercero, calcular la distancia a la cual se encontraba el objeto y, cuarto, se movía el robot hacia el frente en la dirección del eje Roll una distancia definida por la siguiente condición, tomando en cuenta que D_A se refiera a la distancia de avance del robot y D_O se refiere a la distancia entre la cámara del robot y el objeto:

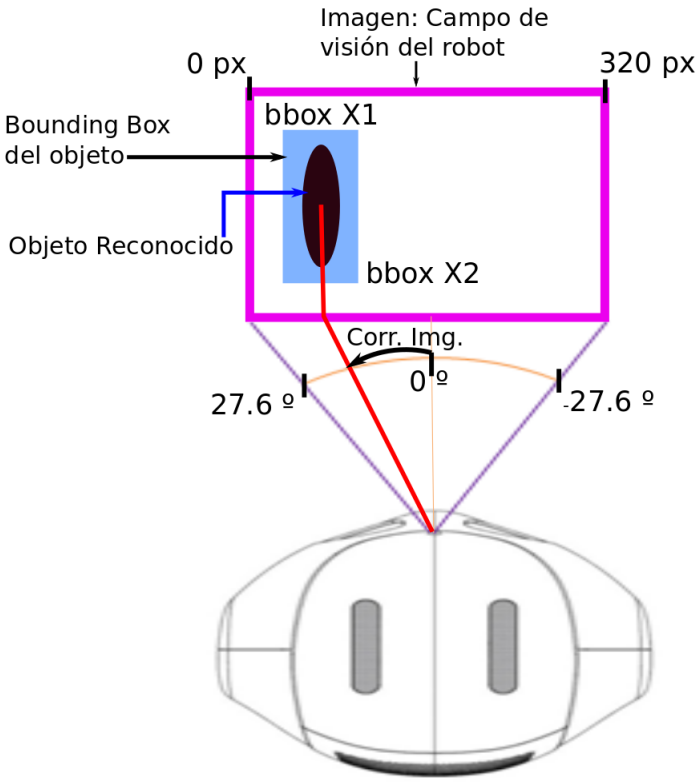


Figura 3.17: Diagrama para el cálculo de corrección de ángulo por imagen (Cabeza del robot).

- Si D_O es mayor a 1.3 metros:

$$D_A = 0.75 D_O \tag{3.4}$$

- Si D_O es menor a 1.3 metros:

$$D_A = 1.00(D_O - 0.97) \tag{3.5}$$

- Si D_O es menor a 1 metro, se detiene este proceso iterativo, ya que significa que el robot ya está aproximadamente a un metro del objeto.

Este proceso iterativo se muestra en la imagen de la figura 3.18, donde las líneas roja muestran la trayectoria ideal en la que debía avanzar el robot y las líneas discontinuas azules muestran las trayectorias reales del robot conforme se acercaba al objeto. El círculo verde indica el objeto de interés.

Para demostrar la necesidad de este procedimiento iterativo, en las siguientes imágenes, se observa la forma en que el objeto se desviaba de la cámara del robot conforme este se acercaba. La primera etapa era mapear la zona en busca del objeto de interés, tal como se muestra en la figura 3.19.

Una vez que se localizaba el objeto, lo siguiente fue acercar el robot al objeto. En las próximas imágenes, mostradas en la figura 3.20 para el primer acercamiento y en la figura

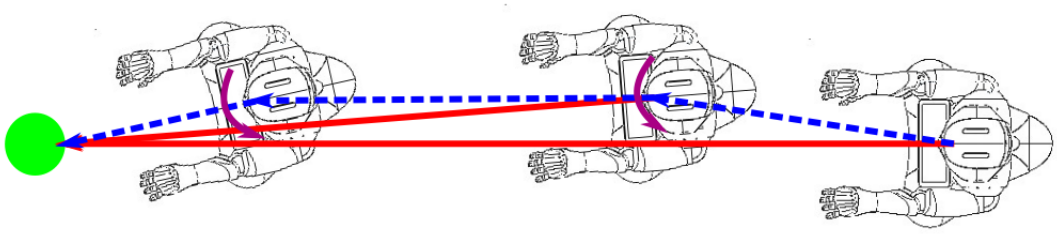


Figura 3.18: Proceso iterativo de acercamiento del robot Pepper al objeto.

(a) Campo de visión del robot mediante imagen RGB.

(b) Objeto reconocido indicado por el *bounding box*.

(c) Cámara del robot centrado con el objeto.

Figura 3.19: Etapa: Mapeo del entorno.

3.21 para el segundo acercamiento, se muestra que, durante la etapa de acercamiento, el objeto se desviaba del centro vertical del campo de visión del robot, lo cual indica que fue necesario y certero haber implementado este proceso iterativo para corregir la dirección de desplazamiento del robot.

Al observar las imágenes en la figura 3.21, se evidenció un problema, conforme el robot se acercaba al objeto, este se salía del campo de visión del robot. Para solventar este problema, se aplicó una relación lineal, la cual estableció que, conforme el robot se acercaba al objeto, según la distancia a la que se encontraba el mismo, se debía cambiar el ángulo de la articulación HeadPitch, encargada de rotar la cámara del robot en la dirección vertical.

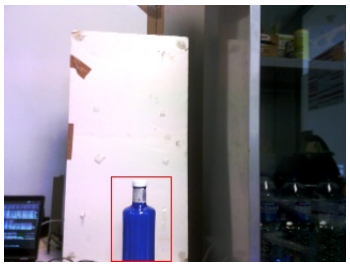
		Distancia al objeto (m)	Ángulo HeadPitch (°)
Ítem	1	5	0
	2	1	16

Tabla 3.2: Coordenadas para cálculo de relación lineal entre la distancia al objeto y el cambio en la articulación HeadPitch.

El cálculo de la función lineal que pasa por esas dos coordenadas, con los datos de la tabla 3.2, se muestra en la ecuación 3.6, donde $\angle H_{pitch}$ se refiere al ángulo de la articulación HeadPitch.

(a) Campo de visión del robot mediante imagen RGB.

(b) Campo de visión del robot mediante mapa de profundidad.



(c) Objeto reconocido indicado por el *bounding box*.

(d) Camara del robot centrado con el objeto.

Figura 3.20: Etapa: Acercamiento del robot al objeto, iteración 1.

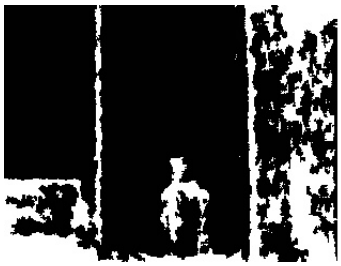
$$\angle H_{pitch} = -4D_O + 20 \tag{3.6}$$

Seguidamente, se muestra otra prueba realizada de acercamiento hasta el objeto, esta vez implementando la corrección de la articulación HeadPitch. En la figura 3.22, se observa la etapa de mapeo del entorno. En las próximas imágenes, en la figura 3.23, figura 3.24 y figura 3.25, se observan las iteraciones del acercamiento del robot hacia el objeto. Es claro que haber implementado esta corrección del ángulo HeadPitch mejoró la ubicación del objeto dentro del campo de visión del robot, elevando las probabilidades de que el objeto fuera reconocido.

3.9 Elección de objetos

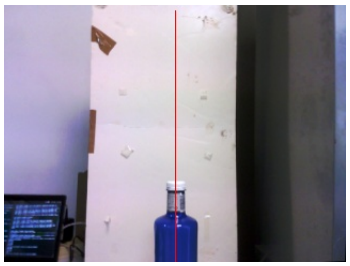
En este punto del proyecto, se determinó la necesidad de seleccionar los objetos definitivos que el robot debía buscar, ya que la siguiente etapa era la de sujetar el objeto. El procedimiento para el entrenamiento fue el mismo de la sección 3.4, con la diferencia de que fueron utilizados más datos de entrenamiento. Las clases de entrenamiento fueron una botella de agua, una caja de café, un recipiente de chocolate y una caja de palomitas, mostrados en la figura 3.26, además de una clase de rostros faciales, que al final del proyecto no se utilizó, la razón se explicará más adelante.

(a) Campo de visión del robot mediante imagen RGB.



(b) Campo de visión del robot mediante mapa de profundidad.

(c) Objeto reconocido indicado por el *bounding box*.

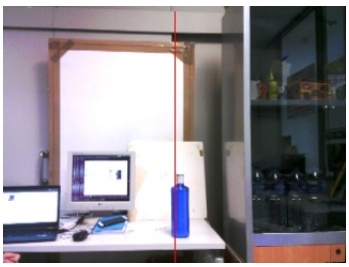


(d) Camara del robot centrado con el objeto.

Figura 3.21: Etapa: Acercamiento del robot al objeto, iteración 2.

(a) Campo de visión del robot mediante imagen RGB.

(b) Objeto reconocido indicado por el *bounding box*.



(c) Cámara del robot centrado con el objeto.

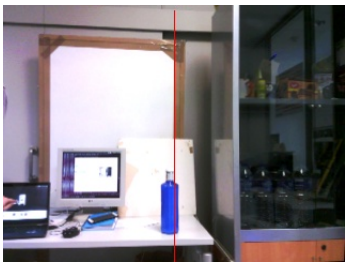
Figura 3.22: Etapa: Mapeo del entorno, corrección de HeadPitch implementado.

Este entrenamiento se realizó utilizando el modelo *multi-class* de Faster R-CNN en Caffe, durante 50 000 iteraciones. Se utilizó el *solver* denominado *Descenso por Gradiente Estocástico con Nesterov*, con un *learning rate* de 0.001 y un *momentum* de 0.9. Para la etapa de prueba, se observó que hubo un sobre entrenamiento para la categoría de palomitas, a partir de la iteración 10 000. Esto, probablemente, fue causado porque el conjunto de datos para esta categoría no fue preparado correctamente, por razones como la calidad de la imagen o selección del *ground truth*. Una vez listos los objetos definidos y la red entrenada, se pudo continuar con el proceso de sujetarlos y etapas posteriores.

(a) Campo de visión del robot mediante imagen RGB.

(b) Campo de visión del robot mediante mapa de profundidad.

(c) Objeto reconocido indicado por el *bounding box*.



(d) Cámara del robot centrado con el objeto.

Figura 3.23: Etapa: Acercamiento del robot al objeto, iteración 1, corrección de HeadPitch implementado.

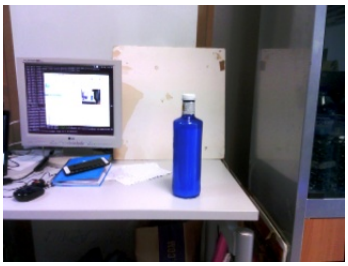
3.10 Sujetar el objeto con los brazos del robot Pepper

Una vez que el robot fue capaz de acercarse al objeto deseado a un metro de distancia, se buscó la forma de que el robot se acercara a una distancia de 0.4 metros del objeto. Debido a que la cámara 3D no funciona a esa distancia, se propuso determinar la distancia a la cual se encontraba el objeto mediante las dimensiones del *bounding box* determinado por la red neuronal.

El experimento consistió en detectar el objeto botella y obtener las dimensiones de su *bounding box* a dos distancias desde el robot, primero a 0.97 metros y luego a 2.9 metros. Cuando se obtuvo esta información, se aplicó relación lineal entre la distancia entre el objeto y el robot, en función de la longitud horizontal y vertical del *bounding box*.

Utilizando los datos de la tabla 3.3, la primera relación lineal que se obtuvo fue entre la distancia horizontal del *bounding box* y la distancia entre el robot y el objeto, mostrada en la ecuación 3.7, donde $BBOX_X$ se refiere a la longitud del *bounding box* en la dirección horizontal.

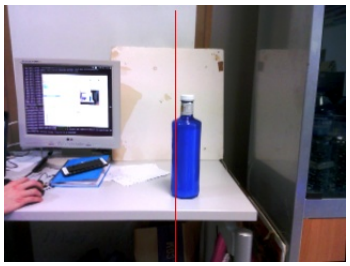
$$D_O = 5.216 - 0.0772 BBOX_X \tag{3.7}$$



(a) Campo de visión del robot mediante imagen RGB.

(b) Campo de visión del robot mediante mapa de profundidad.

(c) Objeto reconocido indicado por el *bounding box*.



(d) Camara del robot centrado con el objeto.

Figura 3.24: Etapa: Acercamiento del robot al objeto, iteración 2, corrección de HeadPitch implementado.

Distancia horizontal de *bounding box* (px)	Distancia entre objeto y robot (m)
55	0.97
30	2.90

Tabla 3.3: Datos para calcular la relación lineal entre la distancia horizontal del *bounding box* y la distancia entre el robot y el objeto.

Utilizando los datos de la tabla 3.4, la otra relación obtenida fue entre la distancia vertical del *bounding box* y la distancia entre el robot y el objeto, mostrada en la ecuación 3.8, donde $BBOX_Y$ se refiere a la medida del *bounding box* en la dirección vertical.

Distancia vertical de *bounding box* (px)	Distancia entre objeto y robot (m)
108	0.97
55	2.90

Tabla 3.4: Datos para calcular la relación lineal entre la distancia vertical del *bounding box* y la distancia entre el robot y el objeto.

$$D_O = 4.9028 - 0.0364 BBOX_Y \tag{3.8}$$

(a) Campo de visión del robot mediante imagen RGB.

(b) Campo de visión del robot mediante mapa de profundidad.

(c) Objeto reconocido indicado por el *bounding box*.

(d) Camara del robot centrado con el objeto.

Figura 3.25: Etapa: Acercamiento del robot al objeto, iteración 3, corrección de HeadPitch implementado.

Posteriormente, en los resultados, se expone la razón por la cual este método no fue viable, debido a la inestabilidad del *bounding box* calculado por la red convolucional.

Una vez que el robot se encontraba a un metro de distancia del objeto, lo siguiente fue idear un método para que el robot sujetara el objeto. Para lograrlo, primero se implementó una secuencia de preparación de los brazos, con el fin de evitar que golpeara la mesa donde se encontraba el objeto. Idealmente, para colocar los brazos del robot en una coordenada en el espacio, se tuvo que haber utilizado el método de cinemática inversa, pero, por falta de tiempo, en este proceso se utilizó cinemática directa, donde se le definió el ángulo al cual se debía colocar cada articulación del brazo.

Un serio problema que se presentó fue que los brazos del robot alcanzaban en un tiempo aproximado de 5 minutos temperaturas de 65 ° C que dificultaban su correcto funcionamiento y, por lo tanto, para enfriar los motores se debía colocar el robot en posición de reposo por, aproximadamente, 45 minutos, lo cual significó pérdida de tiempo valioso para realizar más pruebas con el robot. A pesar de estas complicaciones, en la tabla 3.5, se muestra la secuencia de ángulos para preparar los brazos del robot, utilizada cuando se encontraba a un metro de distancia de cualquier objeto.

Una vez que los brazos del robot se encontraban en posición, se tuvo que mover la base del robot hacia adelante, en dirección del eje principal *Roll*, una distancia específica para cada objeto mostrado en la tabla 3.6.

(a) Caja de Palomitas.

(b) Caja de Café.

(c) Botella.

(d) Recipiente de Chocolate.

Figura 3.26: Objetos definitivos que debe reconocer el robot.

Secuencia de movimiento	Articulación	Ángulo (°)	Articulación	Ángulo (°)
1	LShoulderPitch	-10	RShoulderPitch	-10
2	LElbowRoll	-60	RElbowRoll	60
3	LShoulderRoll	40	RShoulderRoll	-40
4	LElbowYaw	0	RElbowYaw	0
5	LWristYaw	-90	RElbowYaw	90

Tabla 3.5: Posición para preparar los brazos y acercarse.

Cuando el robot se encontraba en posición para tomar el objeto, se le indicó una secuencia de movimientos aplicados, mediante cinemática directa, a las articulaciones de los brazos del robot, según el objeto en cuestión, tal como se muestra en la tabla 3.7 para la botella, tabla 3.8 para la caja de café, tabla 3.9 para el recipiente de chocolate y tabla 3.10 para la caja de palomitas.

Una vez sujetado el objeto, se le indicó al robot levantar ligeramente los brazos, luego desplazar su base hacia atrás en dirección del eje principal *Roll* y, por último, bajar

Objeto	Distancia a avanzar en eje roll (m)
Botella	0.75
Caja de café	0.81
Recipiente de chocolate	0.80
Caja de palomitas	0.78

Tabla 3.6: Distancia de avance del robot para sujetar los objetos.

Posición de movimiento	Articulación	Ángulo (°)
1	LHand	1
2	RHand	1
3	LShoulderRoll	18
4	RShoulderRoll	-18
5	LShoulderRoll	10
6	RShoulderRoll	-10

Tabla 3.7: Secuencia de movimientos para sujetar la botella.

de nuevo los brazos para que no interfieran con su campo de visión. Esta secuencia de movimientos se observa en la tabla 3.11.

3.11 Llevar el objeto al paciente

Una vez que el robot sujetaba el objeto, se debía dirigir hasta el punto donde se encontrara el paciente. El método para realizar esta tarea fue exactamente el mismo de buscar un objeto en el entorno, excepto por la condición de que ahora el objeto que debía buscar era la cara del paciente o un punto de referencia, por ejemplo, un código QR. Para el primer caso, si se utilizaba la categoría de rostro de la última red entrenada, existía el riesgo de que detectara un rostro que no fuera el del paciente, sino el de otra persona en la habitación o incluso una fotografía de un rostro humano, lo cual podía confundir al robot.

Para solucionar esto, esa misma red se pudo haber entrenado, con el fin de que detectara rostros de personas específicas, pero había altas probabilidades de que, incluso teniendo una red que diferenciara entre rostros, igual cometiera errores, debido a un mal entrenamiento a causa de un conjunto de datos mal preparado. Por esta razón, se decidió que el punto donde el robot debía llevar el objeto estaría definido por un marcador del tipo código QR.

Aunque este reconocimiento de código QR se lleva a cabo por otro método diferente a CNN, la lógica de búsqueda del código es exactamente la misma, porque ROS facilitó que

Posición de movimiento	Articulación	Ángulo (°)
1	LHand	1
2	RHand	1
3	LShoulderRoll	18
4	RShoulderRoll	-18
5	LShoulderRoll	14
6	RShoulderRoll	-14

Tabla 3.8: Secuencia de movimientos para sujetar la caja de café.

Posición de movimiento	Articulación	Ángulo (°)
1	LHand	1
2	RHand	1
3	LShoulderRoll	18
4	RShoulderRoll	-18
5	LShoulderRoll	13
6	RShoulderRoll	-13

Tabla 3.9: Secuencia de movimientos para sujetar el recipiente de chocolate.

el servicio de reconocimiento de códigos QR tuviera exactamente la misma estructura que la respuesta del servicio de clasificación de objetos. El servicio de cálculo de profundidad del objeto dentro de un bounding box se mantuvo intacto. De esta manera, cuando el robot detectaba el código QR, este se debía desplazar hasta donde se ubicaba el código QR, mientras sujetaba el objeto. Cuando el robot detectaba que se encontraba a un metro del código QR, daba inicio el procedimiento de colocar el objeto en frente del código, al lado de donde se encontraba el paciente. En la tabla 3.12, se muestra la secuencia de movimientos usada para lograr colocar los objetos respectivamente sobre la mesa al lado del paciente.

De esta manera, se demostró el funcionamiento de cada fase del proyecto junto con la forma en que se enlazaron, para lograr un funcionamiento completo entre estos. En la figura 3.27, se muestra el diagrama final que describe la lógica de funcionamiento de este proyecto.

Aquí culmina la descripción de la solución para afrontar el problema. Seguidamente, se muestran los resultados que surgen de esta etapa junto con su respectivo análisis.

Posición de movimiento	Articulación	Ángulo (°)
1	LHand	1
2	RHand	1
3	LShoulderRoll	18
4	RShoulderRoll	-18
5	LShoulderRoll	11
6	RShoulderRoll	-11

Tabla 3.10: Secuencia de movimientos para sujetar la caja de palomitas.

Secuencia de movimiento	Articulación	Ángulo (°)	Articulación	Ángulo (°)
1	LShoulderPitch	10	RShoulderPitch	-10
2	Desplazar base 1.2m hacia atrás. Dirección eje *roll*			
3	LShoulderPitch	40	RShoulderPitch	-40

Tabla 3.11: Secuencia para preparar el cuerpo del robot y buscar el código QR.

Secuencia de movimiento	Articulación	Ángulo (°)	Articulación	Ángulo (°)
1	LShoulderPitch	10	RShoulderPitch	-10
2	Desplazar base 0.55m hacia adelante. Dirección eje *roll*			
3	LShoulderPitch	5	RShoulderPitch	5
4	LShoulderRoll	50	RShoulderRoll	-50
5	Desplazar base 0.6 hacia atrás. Dirección eje *roll*			
2	Colocar el robot en posición de descanso (Predefinido por Naoqi)			

Tabla 3.12: Secuencia para soltar el objeto en frente de código QR.

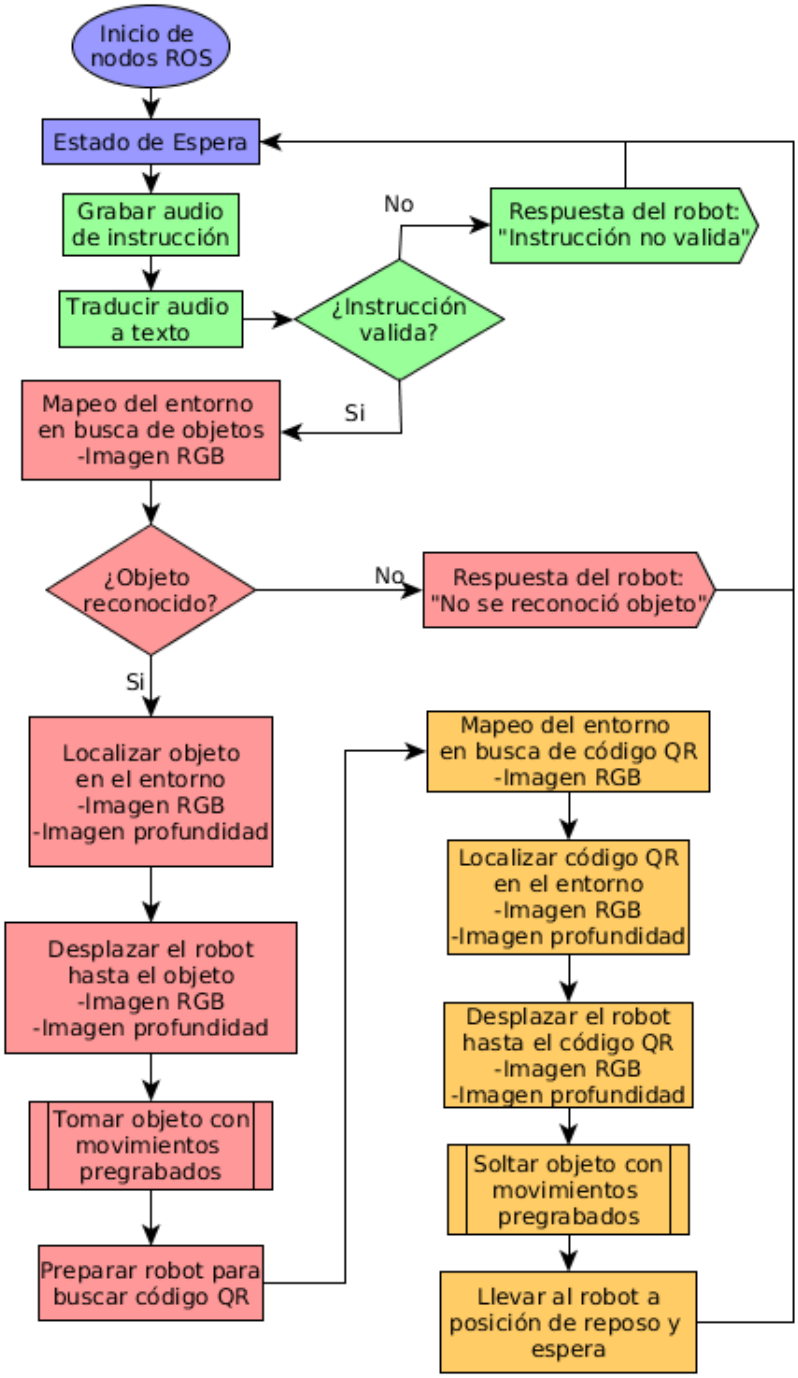


Figura 3.27: Diagrama final de funcionamiento.

Capítulo 4

Resultados y análisis

Este capítulo se deriva de la solución implementada en este proyecto, con el fin de evaluar las condiciones sobre las cuales se lograron los objetivos propuestos. En cada etapa de este proyecto, el uso efectivo de cada herramienta significó un resultado que benefició el funcionamiento del mismo, ya que, si alguna de estas no funcionaba, no se hubiera podido completar el objetivo principal.

Para la primera aproximación de la distancia desde el robot hasta el objeto propuesto en la sección 3.2, el resultado del algoritmo implementado se observa en la figura 4.1, donde se muestra, en la terminal, la distancia promedio a la que se encuentra el objeto y un recorte del mapa de profundidad delimitado por el *bounding box* extraído de la detección del objeto en la imagen RGB.

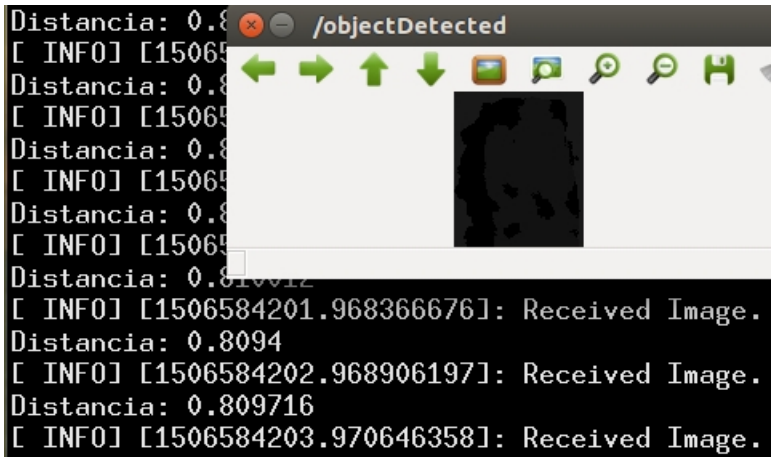


Figura 4.1: Resultado del cálculo de distancia promedio a la que se encuentra un objeto.

Como se mencionó previamente, para esta primera aproximación, se aplicó una media aritmética. Para comprobar que este método funcionó correctamente, se midió la distancia entre el objeto y el Kinect, tal como se muestra en la figura 4.2. La medición marca una distancia de aproximadamente 75 centímetros. A pesar de que hubo una diferencia de 5 centímetros, esto no afectó al sistema, ya que esta herramienta únicamente se utilizó

para que el robot se acercara al objeto aproximadamente a un metro, porque los mapas de profundidad tienen un rango de funcionamiento entre 50 centímetros y 4.5 metros para el modelo Kinect versión 1.0 utilizado en esta prueba [34]. Es importante aclarar que la distancia promedio calculada con el Kinect es mayor, porque los puntos dentro del *bounding box* del objeto, que corresponden a puntos detrás de este, provocaban que la media aritmética de estos puntos incrementara, alejando virtualmente al objeto del Kinect.

Figura 4.2: Medición real del objeto utilizando una cinta métrica.

Una modificación que se tomó en cuenta fue que la cámara 3D que posee el robot era distinta (modelo ASUS XTION), donde el rango de funcionamiento es de 80 centímetros a 3.5 metros [35]. Por esta razón, aunque el reconocimiento de objetos se planteó con un alcance de 5 metros, tal como está escrito en la sección 1.3, el sistema completo fue restringido a que los objetos debían estar en un radio de máximo 3.5 metros.

Una vez comunicados los dos servicios de clasificación de objetos y cálculo de profundidad, tal como se muestra en la figura 3.11, se realizó una prueba utilizando el entorno mostrado en la figura 3.10, donde la distancia medida entre el Kinect y el monitor fue de 88 centímetros. Al evaluar el error con este sistema implementado, se observó que la diferencia entre el valor real y el calculado fue de 1.26 cm, lo cual logró satisfacer las necesidades de esta etapa.

En la sección 3.8, es importante indicar que, en la ecuación 3.4 y la ecuación 3.5, el valor de 97 centímetros en la variable de acercamiento se definió así, para que el robot cumpliera la condición de que el objeto se posicionara a una distancia de un metro. Esta decisión se tomó con base en los resultados mostrados en la tabla 4.1, este experimento consistió

en probar distintos valores de variable de acercamiento y luego medir la distancia entre el robot y el objeto, utilizando el instrumento de medición con tecnología ultrasónica marca STANLEY, modelo 77-007 IntelliMeasure. En la práctica, se observó que, si se le asignaban 97 centímetros (en la ecuación 0.97) a la variable de acercamiento, este valor era el que mejor se ajustaba para que el robot se detuviera a un metro del objeto.

Variable de acercamiento (m)	Distancia hasta el objeto (m)
0.95	0.97
0.96	0.98
0.97	1.00 (toma 2 iteraciones)
0.98	1.00 (toma 10 iteraciones)
0.99	1.01 (No alcanza 1 m)

Tabla 4.1: Pruebas para variable en acercamiento hasta el objeto a un metro de distancia.

En la sección 3.10, para evaluar la viabilidad de aproximar la distancia desde el robot hasta el objeto, utilizando las dimensiones del *bounding box*, en una evaluación se utilizó como elemento de prueba otro *bounding box* del mismo objeto, con dimensiones de 32 px en la dirección horizontal y 64 px en la dirección vertical, donde la distancia medida del objeto fue 1.78 metros.

Al evaluar la distancia horizontal y vertical de este *bounding box* en la ecuación 3.7 y ecuación 3.8, respectivamente, se observó en la tabla 4.2, en la columna llamada Diferencia de Distancias, que la diferencia entre la distancia obtenida de la medición y la calculada era muy grande como para tomarlo como una aproximación correcta.

Dimensión	Valor (px)	Distancia medida (m)	Distancia calculada (m)	Diferencia de distancias (m)
Ancho	32	1.78	2.7456	0.9656
Altura	64	1.78	2.5732	0.7932

Tabla 4.2: Evaluación de aproximación de distancia en base a dimensiones de *bounding box.*

Por esta razón, se decidió que, cuando el robot se encontrara a un metro del objeto, debía acercarse a 40 centímetros del mismo a "ciegas", en el sentido de que nada más se le indicara que se moviera hacia adelante en la dirección del eje *Roll*. Esta solución cumplió con las necesidades del proyecto, tal como se muestra en la tabla 4.3. Para el objeto botella hubo fallos porque, al ser transparente, este permitía que parte del patrón de luz infrarroja generado por el emisor de la cámara XTION, que interactuaba con el material de la botella, no llegara al receptor, lo cual se traducía como una pérdida de información del mapa de profundidad e inducía a un cálculo incorrecto de la profundidad a la que se encontraba la botella.

Objeto	# Pruebas	# Acercamientos exitosos	#Acercamientos fallidos
Botella	10	8	2
Caja de café	10	10	0
Recipiente de chocolate	10	10	0
Caja de palomitas	10	10	0

Tabla 4.3: Evaluación del acercamiento a una distancia de 0.4 metros entre el robot y el objeto deseado.

Uno de los inconvenientes que se presentaron a esa altura del proyecto fue que la distancia calculada por el robot tenía errores de ±5 cm, lo cual provocaba errores a la hora de que el robot tomaba el objeto. Por esta razón, se propuso otro método para mejorar el cálculo de la distancia del objeto. Se debe recordar que, en los mapas de profundidad, cada píxel tiene una profundidad asociada.

Un mejor algoritmo para el cálculo de la distancia se basa en la figura 4.3, donde la unidad de medición de los rangos es metros y cada asterisco significa 10 puntos válidos de profundidad. En dicha figura, para el objeto identificado, se registran 192 puntos mostrados por los asteriscos azules y los demás puntos representan el fondo mostrado por los asteriscos rojos. Por lo tanto, la forma en que se discrimina entre el objeto deseado y los demás objetos es definiendo un *threshold*, de forma tal que, si se supera este *threshold*, se inicia a promediar las distancias de los puntos sobre este y cuando cae debajo de este *threshold*, el conteo se detiene.

Para definir dicho *threshold*, se hicieron 30 pruebas suponiendo un porcentaje entre el 5% y el 15% de la cantidad total de puntos válidos dentro del *bounding box*. Continuando con este ejemplo, la cantidad de puntos válidos es 532. El *threshold* que mejor resultados dio fue el 6.5% de 532 puntos, es decir, aproximadamente 35 puntos. Es así como se logró una mejor aproximación de la distancia a la cual se encontraba el objeto.

Los resultados cuantitativos relevantes para el cumplimiento de los objetivos se muestran a continuación.

La primera prueba realizada evaluó la eficiencia del reconocimiento de objetos utilizando la cámara del robot. Esta prueba consistió en colocar los cuatro objetos deseados dentro del campo de visión del robot, en diferentes entornos, para tomar fotografías de ellos en un rango de un metro a cinco metros, con un paso de 0.5 metros. El resultado de esta prueba se muestra en la tabla 4.4, que indica la cantidad de aciertos por cada objeto y la cantidad de tomas que fueron realizadas, además, consecuentemente, en la tabla 4.5 se indican los mismos resultados en porcentajes de acierto por cada objeto.

Es evidente que el objetivo de reconocer objetos a cinco metros de distancia no se cumplió, ya que la calidad de la cámara del robot modelo ASUS XTION era bastante mala, lo cual provocó pérdida de información, incluso teniendo una R-CNN robusta. De forma similar, se realizó una prueba para comprobar si el reconocimiento de voz era efectivo a un metro de

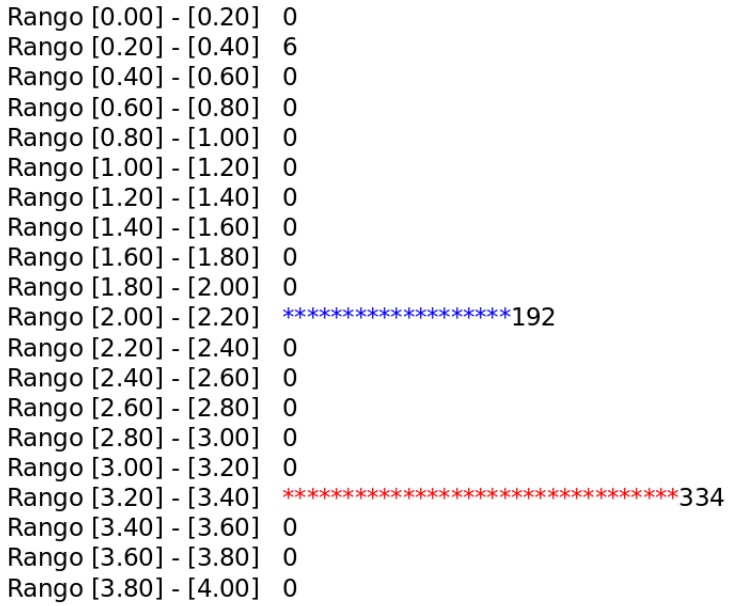


Figura 4.3: Segunda aproximación para el cálculo de profundidad de los objetos utilizando un *threshold* de 6.5%.

Distancia (m)	Cantidad de tomas	Cantidad de aciertos por objeto			
		Botella	Café	Chocolate	Palomitas
1.0	30	29	30	30	11
1.5	30	27	30	22	10
2.0	24	15	17	3	1
2.5	28	17	17	2	0
3.0	29	6	21	7	1
3.5	29	2	2	0	0
4.0	28	1	1	0	0
4.5	28	0	0	0	0
5.0	26	0	0	0	0

Tabla 4.4: Aciertos por objetos utilizando la red Faster R-CNN.

distancia desde el paciente hasta el micrófono, el cual fue colocado en sus cercanías. Esta prueba consistió en decir cinco frases, una activa el *trigger* y cuatro que correspondieron a la cantidad de objetos en estudio, en un rango de 0.5 metros a 1.5 metros, con un paso de 0.5 metros, como se muestra en la tabla 4.6. Al observar los resultados, se puede afirmar que se cumplió el objetivo de reconocer las instrucciones de voz del paciente a un metro de distancia medido desde el micrófono.

El último objetivo por evaluar fue determinar si se cumplió la integración de tareas para lograr un sistema funcional. Se puede afirmar que este objetivo se cumplió satisfactoriamente, aunque con ciertas limitaciones. La primera limitación fue la distancia a la cual se

Distancia (m)	Porcentaje de aciertos por objeto (%)			
	Botella	Café	Chocolate	Palomitas
1.0	96.67	100	100	36.67
1.5	90	100	73.33	33.33
2.0	62.5	70.83	12.5	4.17
2.5	60.71	60.71	7.14	0
3.0	20.68	72.41	24.14	3.45
3.5	6.90	6.90	0	0
4.0	3.57	3.57	0	0
4.5	0	0	0	0
5.0	0	0	0	0

Tabla 4.5: Porcentaje de aciertos por objetos utilizando la red Faster R-CNN.

Distancia (m)	Cantidad tomas	Cantidad de aciertos	Porcentaje de aciertos (%)
0.5	10	10	100
1	10	10	100
1.5	10	9	90

Tabla 4.6: Aciertos de frases utilizando reconocimiento de voz.

lograron reconocer los objetos, además de otros factores como la orientación del objeto, el entorno que lo rodeaba, iluminación y calidad de la cámara del robot. La segunda limitación fue a nivel de movimiento de articulaciones de los brazos del robot, los cuales responden a las señales enviadas de manera fortuita, donde la principal explicación fue que los motores se calentaban y dejaban de funcionar, debido a un sistema de protección integrado en el robot.

En el siguiente capítulo, se propone el trabajo pendiente para mejorar el desempeño del robot y la eficiencia para cumplir el objetivo de asistir a personas con discapacidad.

Capítulo 5

Conclusiones y recomendaciones

El principal aporte que propuso este proyecto fue demostrar que es posible crear un robot asistente para personas con discapacidad, en busca de garantizarles una mejor calidad de vida y que, debido a que fue un prototipo reciente, el objetivo principal se cumplió bajo las condiciones ya mencionadas. También, es importante destacar que este proyecto expone problemas que se presentan cuando los proyectos se desarrollan en la vida real y no solo se quedan a nivel de simulación. Esto representa un gran aporte para los demás investigadores que quieran profundizar en este tipo de proyectos.

Los aportes secundarios contemplan la integración de *deep learning*, reconocimiento de voz y robótica mediante ROS, crear un modelo basado en el proyecto Faster R-CNN, capaz de detectar cuatro objetos y probar que el robot Pepper no está diseñado para esta tarea, aunque sí es capaz de completarla. Es importante destacar que el trabajo mostrado en este documento es la base de un proyecto mayor, en el cual es necesario progresar a nivel de construcción del robot y programación robusta.

A nivel de recomendaciones para trabajos futuros, en el tema de clasificación de objetos, se aconseja entrenar otra red con un conjunto de datos de los objetos amplio y variado en temas de entorno, iluminación y puntos de vista, con la meta de generar un modelo de red neuronal robusto para incrementar los porcentajes de aciertos obtenidos en este proyecto. Además, otra contribución sería si el nuevo modelo entrenado logra detectar más objetos de los propuestos en este trabajo. Otro sistema que le añadiría mucho peso al proyecto es que logre diferenciar entre el paciente y el resto de las personas y fotografías en el entorno, para prescindir del código QR.

Otra sugerencia para mejorar este proyecto es que, si se continúa utilizando el robot Pepper, es necesario desarrollar un método que calcule la cinemática inversa de los brazos del robot, para facilitar su manipulación y, eventualmente, integrar la herramienta de grasping. Para reforzar la propuesta de mantener el diseño original de Pepper, se recomienda incorporar un sistema de navegación que le permita al robot ubicarse dentro del entorno que lo rodea e identificar objetos importantes dentro de este entorno, tales como los objetos, personas y obstáculos.

En caso de que se plantee reestructurar el robot, un buen aporte sería cambiar la cámara 2D para que tenga una mejor resolución de imagen y cambiar la cámara 3D con un rango de alcance de 0.3 metros a 5.5 metros y, en caso de que esta última recomendación no se lograra, se debe proponer otra técnica para calcular la distancia del objeto deseado. También sería adecuado cambiar los brazos robóticos por unos de seis grados de libertad, para lograr mejores resultados en cuestión de posición, orientación y estabilidad del efecto final del brazo.

Bibliografía

[1] Organización Mundial de la Salud. Informe mundial la discapacidad. 2011.

[2] M. Cazorla, D. Viejo, and S. Orts. Retogar: Retorno al hogar. sistema de mejora de la autonomía de personas con daño cerebral adquirido y dependientes en su integración en la sociedad. URL http://www.rovit.ua.es/retogar-retorno-al-hogar-sistema-de-mejora-de-la-autonomia-de-personas-con-dano-cerebral-adquirido-y-dependientes-en-su-integracion-en-la-sociedad/.

[3] A. Gardecki and M. Podpora. Experience from the operation of the pepper humanoid robots. In *Progress in Applied Electrical Engineering (PAEE), 2017*, pages 1–6. IEEE, 2017.

[4] G. Runger and D. Mongomery. Probabilidad y estadística aplicada a la ingeniería, 2003.

[5] R. Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[6] M. Copeland. What's the difference between artificial intelligence, machine learning, and deep learning?, July 2016. URL https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/.

[7] Opencv Dev Team. Mat - the basic image container. URL https://docs.opencv.org/2.4/doc/tutorials/core/mat_the_basic_image_container/mat_the_basic_image_container.html.

[8] D. Hubel and T. Wiesel. Receptive fields of cells in striate cortex of very young, visually inexperienced kittens. *Journal of neurophysiology*, 26(6):994–1002, 1963.

[9] R. Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[10] A. Karpathy. cs231n convolutional neural networks for visual recognition. URL http://cs231n.github.io/neural-networks-1/.

[11] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

[12] P. Veličković. Deep learning for complete beginners: convolutional neural networks with keras. URL https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html.

[13] K.Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[14] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European Conference on Computer Vision*, pages 346–361. Springer, 2014.

[16] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2010.

[17] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[18] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[19] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2553–2561. Curran Associates, Inc., 2013. URL http://papers.nips.cc/paper/5207-deep-neural-networks-for-object-detection.pdf.

[20] P. Arbelaez, M. Maire, and C. Fowlkes. From contours to regions: An empirical evaluation. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2294–2301, June 2009. doi: 10.1109/CVPR.2009.5206707.

[21] C. Gu, P. Arbelaez, and J. Malik. Recognition using regions. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1030–1037, June 2009. doi: 10.1109/CVPR.2009.5206727.

[22] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[23] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. Berg, and L. Fei-Fei. ImageNet Large Scale Visual

Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3): 211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[24] H. Zhan. Detection: Faster r-cnn. URL https://huangying-zhan.github.io/2016/09/22/detection-faster-rcnn.html.

[25] J. Craig. *Robótica*. Pearson Educación de México, 2006. ISBN 9789702607724. URL https://books.google.es/books?id=hRzOp_qdxG8C.

[26] SoftBank Robotics. Pepper - technical overview, 2017. URL http://doc.aldebaran.com/2-4/home_pepper.html.

[27] ROS. Ros documentation, Enero 2017. URL http://wiki.ros.org/.

[28] A. Bernin. Kinect chess board meta pattern. URL http://livingplace.informatik.haw-hamburg.de/blog/?p=229.

[29] E. Morales and E. Sucar. Reconocimiento de voz, Enero 2017. URL http://ccc.inaoep.mx/~esucar/Clases-ia/Laminas2017/recvoz.pdf.

[30] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[31] R. Girshick. Faster r-cnn (python implementation). URL https://github.com/rbgirshick/py-faster-rcnn.

[32] A. Zhang. Speech recognition module for python. URL https://github.com/Uberi/speech_recognition#readme.

[33] J. Rangel. pepper_kinetic. URL https://github.com/jcarlos2289/pepper_kinetic.

[34] Microsoft. Kinect hardware. URL https://developer.microsoft.com/en-us/windows/kinect/hardware.

[35] ASUSTeK Computer Inc. Xtion. URL https://www.asus.com/es/3D-Sensor/Xtion/specifications/.

Apéndice A

Resumen Técnico del robot Pepper

Altura de construcción	1208.5 mm
Ancho de construcción	477.2 mm
Profundidad de construcción	424 mm
Peso de construcción	27.82 kg
Batería	Tensión Nominal 26.46V Capacidad 30Ah
Tarjeta madre	Procesador ATOM Z530, CLK Speed 1.6 Ghz, CACHE memory 512 KB, RAM 1GB, FLASH memory 2GB, micro SDHC 8GB
Conectividad	WiFi 802.11 a/b/g/m
Cantidad Parlantes	2 parlantes
Cantidad micrófonos	4 micrófonos
Cámaras	2 camaras YUV con capacidad de captura a 640p@30fps o 2560p@1fps y un sensor 3D Asus Xtion que trabaja a 320p@20fps
Sensores	1 Unidad inercial 1 Acelerómetro de 3-ejes 1 Giroscopio de 3-ejes 6 Láser 2D 3 Bumpers 2 Sensor ultrasónicos 29 codificadores rotativos magnéticos Botones y sensores táctiles Tablet

Tabla A.1: Resumen técnico de robot Pepper según el fabricante SoftBank Robotics [26]