UNIVERSIDAD AUTÓNOMA DE CIUDAD JUÁREZ

Instituto de Ingeniería y Tecnología

Departamento de Ingeniería Eléctrica y Computación

MEDICIÓN DE MAGNITUDES DE OBJETOS SIN CONTACTO HACIENDO USO DE REDES NEURONALES

Reporte Técnico de Investigación presentado por:

Rubén Jesús Barraza Pérez 148761

Requisito para la obtención del título de:

INGENIERO EN SISTEMAS COMPUTACIONALES

Asesor:

Dr. Josué Domínguez Guerrero

Ciudad Juárez, Chihuahua 1 de junio de 2022

Ciudad Juárez, Chihuahua, a 1 de Junio de 2022

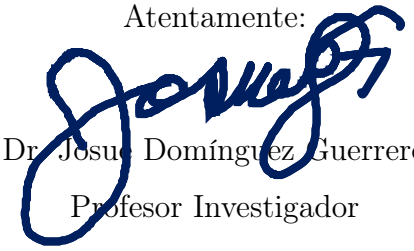
Asunto: Liberación de Asesoría

Mtro. Ismael Canales Valdiviezo
Jefe del Departamento de Ingeniería
Eléctrica y Computación
Presente.-

Por medio de la presente me permito comunicarle que, después de haber realizado las asesorías correspondientes al reporte técnico MEDICIÓN DE MAGNITUDES DE OBJETOS SIN CONTACTO HACIENDO USO DE REDES NEURONALES, del alumno Rubén Jesús Barraza Pérez de la Licenciatura en Ingeniería en Sistemas Computacionales, considero que lo ha concluido satisfactoriamente, por lo que puede continuar con los trámites de titulación intracurricular.

Sin más por el momento, reciba un cordial saludo.

Atentamente:



Dr. Josué Domínguez Guerrero
Profesor Investigador

Ccp: Mtro. David Absalón Uruchurtu Moreno
Coordinador del Programa de Sistemas Computacionales
Rubén Jesús Barraza Pérez
Archivo

Ciudad Juárez, Chihuahua, a 1 de Junio de 2022

Asunto: Autorización de publicación

C. Rubén Jesús Barraza Pérez

Presente.-

En virtud de que cumple satisfactoriamente los requisitos solicitados, informo a usted que se autoriza la publicación del documento de MEDICIÓN DE MAGNITUDES DE OBJETOS SIN CONTACTO HACIENDO USO DE REDES NEURONALES, para presentar los resultados del proyecto de titulación con el propósito de obtener el título de Licenciado en Ingeniería en Sistemas Computacionales.

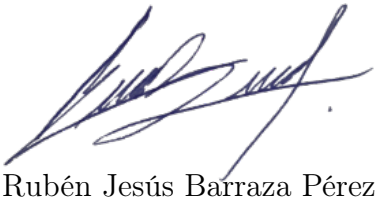
Sin otro particular, reciba un cordial saludo.

Dr. Everardo Santiago Ramírez

Profesor Titular de Seminario de Titulación II

Declaración de Originalidad

Yo, Rubén Jesús Barraza Pérez declaro que el material contenido en esta publicación fue elaborado con la revisión de los documentos que se mencionan en el capítulo de Bibliografía, y que la solución obtenida es original y no ha sido copiada de ninguna otra fuente, ni ha sido usada para obtener otro título o reconocimiento en otra institución de educación superior.



Rubén Jesús Barraza Pérez

Agradecimientos

Primeramente quiero agradecer especialmente a mis padres por el esfuerzo y el apoyo que me dieron, no solo durante la carrera, sino toda la vida. Agradezco a mi asesor Dr. Josué Domínguez Guerrero, que me apoyó con mis dudas en todo momento y me dirigió con sus conocimientos con total profesionalismo brindando su asesoría durante la elaboración de este proyecto. Quiero agradecer de la misma manera al Dr. Everardo Santiago Ramírez pues con sus observaciones enriqueció de manera positiva este documento final.

Dedicatoria

Desde que nací mis padres siempre se aseguraron de ser el pilar de mi vida, me guiaron por el buen camino y me recordaron constantemente que la educación es lo más importante. Por eso quiero dedicar este documento de titulación principalmente a ellos pues no estaría donde me encuentro y no lograría lo que he logrado de no ser por su apoyo y amor incondicional. Dedico este documento a mi madre por siempre estar cuando la necesité, por todos los consejos y todas las enseñanzas de vida. A mi padre por todos los sacrificios que hizo para proveer y darme la educación que tengo. A mi hermana por ser mi apoyo emocional y por enseñarme que no hay imposibles.

Por último le dedico este logro a mis abuelos maternos por toda la sabiduría que me compartieron a lo largo de mi vida, pero especialmente a mi abuela Elodia Alvarado quien siempre deseó que uno de sus nietos varones concluyera sus estudios.

Este logro se lo dedico a todos ellos, agradezco su apoyo y sacrificio y espero este documento sea la muestra de que todo este sacrificio valió la pena.

Resumen

En el presente documento se expone la necesidad de la automatización de la tarea de medición de longitudes de objetos, enfocado en situaciones de no contacto, se propone un prototipo el cual mediante una cámara de profundidad y un equipo de cómputo con un nivel de procesamiento promedio se logra crear un algoritmo que busca cubrir la necesidad de medición de objetos sin contacto. El objetivo de este proyecto fue llegar a una aproximación de las medidas de un objeto apoyándose en las técnicas actuales de aprendizaje profundo con redes neuronales convolucionales para la detección del objeto para posteriormente calcular su largo y su altura de una manera simple. El producto fue un prototipo capaz de detectar un objeto y mostrar su largo y altura en pantalla con una precisión mayor al 80 %.

Palabras claves: Cámara de profundidad, Redes neuronales, Medición de objetos , Intel Realsense.

Índice general

Índice de figuras

Índice de tablas

Introducción

El uso de redes neuronales para la realización de mediciones es un tema que no ha sido estudiado a fondo. La información que se obtiene del entorno está en constante crecimiento y se busca resolver temas más complejos. El automatizar tareas como lo es medir, ha llevado al desarrollo de aplicaciones para dispositivos móviles que sean capaces de medir un objeto con tan solo apuntar la cámara, tal es el caso de *ARuler*, desarrollado por Grymala, el cual emplea realidad aumentada para calcular ángulos, distancias y áreas, o *Telémetro smart measure*, una aplicación que permite conocer la distancia a la que está un objeto y su altura haciendo uso de trigonometría. Los anteriores son algunos ejemplos de cómo esta necesidad puede ser solucionada.

En este documento el problema es abordado utilizando una cámara 3D, una red neuronal para detección automática de objetos y usando la distancia del mismo y la cantidad de pixeles ocupados por el mismo en la imagen aproximar sus dimensiones. El prototipo desarrollado logra una efectividad mayor al 80 % bajo un ambiente controlado.

En el Capítulo 1 de este documento se abordan los conceptos claves del aprendizaje profundo, formatos de imagen y algoritmos de reconocimiento de patrones. Después, en el Capítulo 2 se describen las técnicas, dispositivos y aplicaciones que son útiles para el desarrollo de una solución a la necesidad de medición automática que es desarrollada en el Capítulo 3, exponiendo los resultados obtenidos y llegando a las conclusiones en los apartados finales.

Capítulo 1
Planteamiento del Problema

En este apartado de la investigación se abordan conceptos base para una mejor comprensión, al igual que trabajos relevantes para la realización del proyecto. Además, se define la problemática abordada en este proyecto de titulación y se definen objetivos para la solución de la misma.

1.1. Antecedentes

En [1] se aborda el concepto de aprendizaje profundo como el conjunto de una gran variedad de métodos, como redes neuronales, modelos probabilísticos jerárquicos y una variedad de algoritmos de aprendizaje supervisados y no supervisados. Incluso se mencionan algunos factores que dieron paso al desarrollo de estas técnicas contribuyendo al impulso del aprendizaje profundo, como lo fue la aparición de grandes conjuntos de datos etiquetados, de alta calidad y disponibles públicamente, junto con el desarrollo de la computación GPU (*graphics processing unit*, unidad de procesamiento gráfico) paralela, que permitió la transición de un entrenamiento basado en CPU a uno basado en GPU permitiendo así una aceleración significativa en el entrenamiento de modelos profundos. Con todo este poder de procesamiento se pudo abordar nuevos retos para que las computadoras realizaran tareas que requirieran cierto nivel de inteligencia.

La visión por computadora es uno de los desafíos por conquistar del aprendizaje profundo y surge inspirada en el sistema visual humano, el cual se dice, es la mayor fuente de información para las personas y sugiere el tratamiento de ésta por medio de diferentes técnicas [2]. El dispositivo encargado de la tarea de ver en la visión artificial es una cámara de vídeo (o cualquier otro dispositivo similar), que convierte la energía de la luz en corriente eléctrica (cámaras analógicas) o señales digitales (cámaras digitales) [3]. Esta tecnología abre las puertas a diferentes tipos de aplicaciones, tales como, sistemas de vigilancia automatizados, Televisión digital interactiva (TVDi), entre otros, como se indica en [4].

Diferentes tipos de datos se han utilizado para la construcción de algoritmos de reconocimiento de objetos, como imágenes a blanco y negro. Después se incluyeron en esos algoritmos las imágenes a color *Red, Green, Blue* (RGB) y en los últimos años se han introducido al mercado una variedad de sensores baratos y accesibles [5]. Mientras que la mayoría de los trabajos en aprendizaje profundo se enfocan en imágenes 2D, estudios indican que el uso de datos de profundidad genera mejores resultados al momento de hacer detección de objetos [6]. El formato *Red, Green, Blue, Depth* (RGBD) brinda nuevos datos de profundidad al momento de capturar imágenes [7]. Este formato se volvió popularmente reconocido con el lanzamiento del Microsoft Kinect, el cual, introdujo la detección de movimiento de los cuerpos a los videojuegos. Esto generó grandes cantidades de entusiastas en el consumo de esta tecnología y mayor desarrollo en torno a la misma, tal es el caso de los dispositivos RealSense de Intel que lograron integrar esta tecnología a algunas computadoras de escritorio todo en uno o *all in one* [8].

En [9] se creó una red neuronal convolucional profunda que toma un modelo pre-entrenado con datos 2D, modificado para aceptar datos 3D y lo combina con otros dos modelos especializados en el manejo de datos 3D dando resultados positivos al momento de reconocer objetos con datos de profundidad, quedando cerca de un 92 % de efectividad.

En [6] se utilizaron modelos 3D CAD, renderizados desde diferentes (cientos) puntos de

vista para así generar mapas de profundidad como los creados por los sensores de profundidad, y utilizar cada uno de estos renderizados para entrenar una SVM (*Support Vector Machines*, Máquina de vector de Soporte) que es un algoritmo de clasificación de aprendizaje profundo, por último unen todos los SVM para crear un detector 3D del objeto.

En [10] se utilizaron los algoritmos de reconocimiento de patrones SIFT (*Scale-Invariant Feature Transform*) y SURF (*Speeded Up Robust Features*) para implementar en plataformas robóticas móviles enfocadas en tareas *pick-and-place*. En [11] se aplicó el método de nubes de puntos para su uso en los sistemas de reconocimiento de objetos, utilizando descriptores SURF para datos 2D y VFH (Viewpoint Feature Histogram) para datos 3D, al final, mediante un algoritmo de decisión se obtienen similitudes con descriptores guardados en la base de datos.

1.2. Definición del problema

Para medir el largo y el alto de un objeto generalmente se tiene que hacer físicamente, sin embargo, no siempre es posible estar en contacto con estos objetos, pero se puede obtener una estimación usando cámaras de profundidad.

1.3. Objetivo general

Implementar un algoritmo capaz de detectar y medir el largo y ancho de objetos usando redes neuronales convolucionales y una cámara de profundidad.

1.3.1. Objetivos específicos

- Diseñar un algoritmo que sea capaz de detectar objetos y medir magnitudes.

- Elaborar un programa para importar y preprocesar la red neuronal pre-entrenada.
- Comparar e implementar la arquitectura de redes convolucionales que se adecue más a las necesidades del proyecto.

1.4. Justificación

El hacer un algoritmo que utilice las nuevas tecnologías de las cámaras 3D para la detección y medición de objetos, tiene un impacto académico positivo pues se creará una herramienta de software que pueda implementarse en otros proyectos que lo necesiten para la generación de nuevo conocimiento. También tiene un impacto tecnológico porque la creación de una herramienta de medición solo por medio de cámaras puede generar gran utilidad en diferentes sectores, por ejemplo en los sectores industriales, ya que significaría un aumento de efectividad y seguridad de activos al momento de la inspección de piezas. También genera un avance en las áreas donde el reconocimiento y comprensión de escenarios son esenciales para lograr un fin por ejemplo en el campo de la robótica ayudaría con la interacción en tiempo real con el entorno.

1.5. Alcances y limitaciones

- Este proyecto aborda los siguientes puntos:
  1. Se desarrolló un prototipo que identifica objetos y calcula sus dimensiones.
  2. El proyecto se realizó para que funcione con la cámara *Intel Realsense*.
- Los siguientes puntos no se abordan en este proyecto:
  1. Se dispone de *hardware* especializado para un rápido entrenamiento de la red neuronal.

2. Existe una variedad de conjuntos de datos para utilizarlos en la red neuronal.

Capítulo 2
Marco teórico

En este capítulo se explican los conceptos de la tecnología empleada en este proyecto, herramientas al igual que materiales de apoyo que apoyen su funcionamiento para la aplicación en el prototipo realizado.

2.1. Reconocimiento de objetos

La detección de objetos es un área de investigación en el aprendizaje profundo que busca que las computadoras sean capaces de reconocer cualquier objeto con la facilidad con la que lo hace un ser humano. Ha sido un área de investigación que ha estado activa por décadas [12].

El reconocimiento es una función básica, primordial y compleja de la visión por computadora, por medio de ésta el sistema es capaz de aprender a reconocer las formas para posteriormente clasificarlas de forma correcta [2].

Se le llama reconocimiento de objetos al proceso llevado a cabo en la visión artificial o visión por computadora, donde, el software en conjunto con el hardware interpreta las imágenes captadas para su procesamiento. Esto consta de cuatro etapas [13]:

- Captura: Consiste en la captura de imágenes por medio de un sensor o cámara.
- Análisis: Mediante la aplicación de filtros se descartan partes de la imagen y se enfoca

en las partes con mayor probabilidad de coincidencia con lo buscado.

- Segmentación: Consiste en aislar los elementos de la imagen que se desean analizar, la segmentación permite comprender la imagen individualizando sus elementos.
- Reconocimiento: En ella se distinguen los objetos segmentados, mediante al análisis de ciertas características, que se establecen previamente para diferenciarlos.

Un ejemplo de las aplicaciones del reconocimiento automatizado sería la comparación de radiografías, resonancias magnéticas y tomografías. En el campo de la biología el distinguir entre especies de animales [14], reconocer frutas mediante clasificadores [15] o también puede utilizarse para remover ciertos objetos de una escena [16].

Algunos algoritmos han adquirido mucho renombre en el campo del reconocimiento de objetos como lo es el algoritmo de Canny, destacado por su gran rapidez y efectividad en la tarea. El algoritmo de Canny [17] consiste en evitar la eliminación de bordes relevantes y no detectar bordes que no existen, también procura que la distancia entre el borde localizado y la posición real del mismo debe ser mínima, por último, debe integrar todos los datos obtenidos en una respuesta final par un único borde.

2.2. Aprendizaje Profundo

Desde que fue posible escanear y cargar imágenes médicas a una computadora, los investigadores se centraron en sistemas de análisis automatizado. Para finales de los noventas, las técnicas supervisadas de aprendizaje profundo donde los datos de entrenamiento se usaban para desarrollar un sistema, se volvió muy popular en análisis de imágenes médicas. Incluso, siguen utilizándose en la actualidad [18]. Esta técnica se mejora con el tiempo y a su vez se descubren nuevos métodos en esta área de investigación. En los últimos años, el rápido

desarrollo de técnicas de aprendizaje profundo ha traído nuevos avances en la detección de objetos, por lo cual, ha captado la atención de los investigadores [19].

2.2.1. Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN, por sus siglas en inglés) son algoritmos de aprendizaje automático supervisado para dar a las computadoras la capacidad de "ver". Para ello tienen capas ocultas especializadas con jerarquía, esto se refiere a que las primeras capas pueden detectar líneas (curvas, rectas) mientras que las más profundas detectan características más complejas como el contorno. Un ejemplo de diseño de red convolucional puede apreciarse en la Figura 2.1

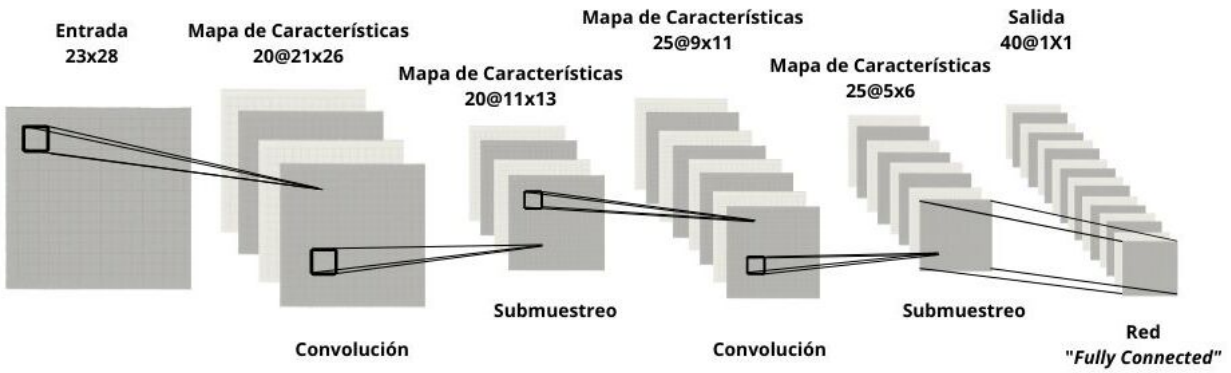


Figura 2.1: Estructura de una red neuronal convolucional típica [20].

Lo que hace diferentes a las CNN's de las redes neuronales tradicionales es el proceso de filtrado de las imágenes o datos antes de introducirlos a las capas de la red, mediante el producto escalar con una matriz de valores predeterminados llamados *kernel's* o núcleos, generando nuevos conjuntos de datos que contienen las características más relevantes de los

originales, a esta parte proceso se le denomina *convolución* [21].

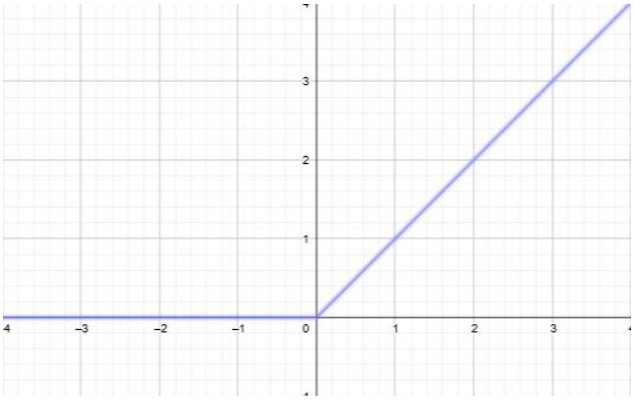
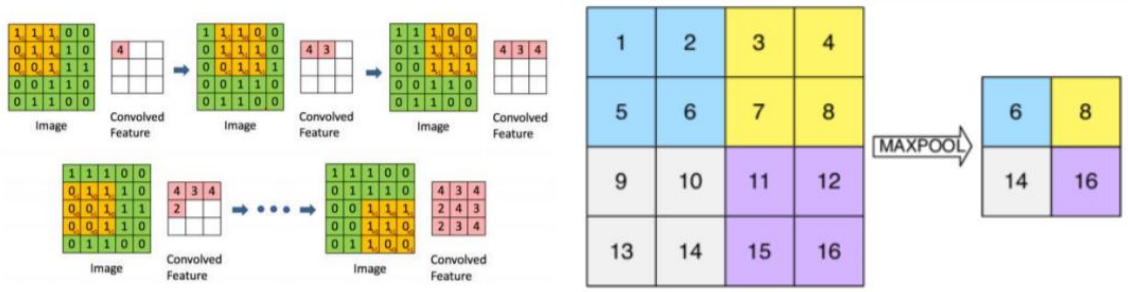


Figura 2.2: Función ReLU.

La salida de ésta se pasa a través de una función de activación la cual es no lineal llamada *ReLU* (Unidad Lineal Rectificada) mostrada en la Figura 2.2, la cual permite un entrenamiento más rápido y con mejores resultados al asignar los valores negativos a cero y manteniendo los valores positivos como se muestra en la Ecuación 2.1.

$$R(z) = max(0, z) \tag{2.1}$$

Después, la dimensión de este mapa de características transformado se reduce mediante agrupación espacial (*Max pooling*), este proceso se puede apreciar en la Figura 2.3. El fin de este último paso es reducir la muestra de una representación de entrada, reducir su tamaño y permitir que el modelo se enfoque en características contenidas en las subregiones agrupadas [22].



(a) Convolución (b) Maxpooling

Figura 2.3: Representación gráfica de una convolución y el resultado de Maxpooling [23].

2.3. YOLO, Detección de Objetos en Tiempo Real

Es un sistema de código abierto que utiliza las CNN para la detección de objetos, su propósito es el de realizar esta tarea en tiempo real. Surgió en el año 2015 con un nuevo enfoque, aplicar una sola red neuronal a toda la imagen reduciendo así el costo computacional y el tiempo de procesamiento. Esta red divide la imagen en regiones y calcula la probabilidad de que exista un objeto en cada una de ellas para identificarlas.

2.3.1. YOLOv4 (Darknet)

Este sistema a tenido diferentes variantes a lo largo de los años desde su publicación. Con el paso del tiempo los desarrolladores fueron agregando actualizaciones al algoritmo para mejorar su precisión y velocidad.

La versión número 4 del sistema desarrollada por Alexey Bochkovskiy descrita en el articulo [24], es un ejemplo de lo mencionado en el párrafo anterior pues las mejoras implementadas incluyen técnicas en reducción de complejidad computacional y entrenamiento a base de datos artificiales para hacer el sistema robusto y tolerante al ruido que puedan tener

las imágenes. Si bien esta red neuronal está preentrenada para detectar 80 clases distintas de objetos gracias al conjunto de datos COCO [25]. Sus desarrolladores han implementado diferentes tipos de configuraciones a este sistema para que se pueda entrenar esta red desde cero, con otro conjunto de datos o mostrar específicos datos de salida.

Su arquitectura general es bastante simple, se basa en un detector en dos fases. La primera fase donde sus cuatro capas interiores etiquetan los píxeles para identificar las regiones donde existen objetos relevantes. La segunda fase es la de depuración de datos, la cual ignora aquella información irrelevante de las regiones clasificadas.

2.4. Cámaras 3D

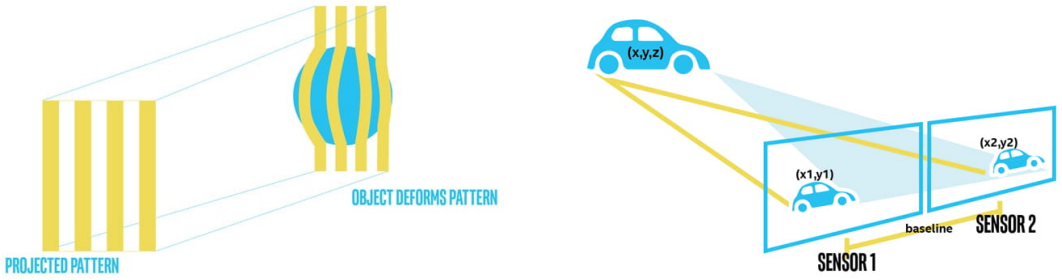
En la tarea de la visión cuando el cerebro necesita más información de la aportada individualmente por cada ojo, utiliza un recurso para interpretar la imagen, la perspectiva. Esta utiliza la memoria visual y aprovecha las deformidades de los objetos y texturas para reconstruirlos en un espacio tridimensional.

Ya que el espacio es tridimensional, es imprescindible conocer las tres direcciones básicas (x, y, z) para tener una interpretación correcta del espacio. La profundidad se hace presente cuando se pueden distribuir los objetos en estas tres direcciones [26].

La cámara 3D también llamada cámara RGB-D es una combinación de sensores que proporciona profundidad y color al momento de capturar imagen, últimamente han adquirido un mayor número de usuarios debido a su bajo precio y a sus aplicaciones en el campo de la robótica y aprendizaje profundo.

Existen varios tipos de cámaras 3D, la clave para diferenciarlas es la manera en que cada una detecta la profundidad. Cada una depende de información conocida para funcionar. Las cámaras de luz estructurada proyectan un patrón de luz infrarroja en la escena y dependiendo

de la deformación de ese patrón, se calcula la profundidad. Las cámaras de profundidad estereoscópica cuentan con dos sensores separados a una distancia conocida que capturan imágenes simultáneamente y las comparan para así generar información de profundidad. Una representación para el funcionamiento de estos dos tipos de cámaras se observa en la Figura 2.4.



(a) Luz estructurada

(b) Profundidad estereoscópica

Figura 2.4: Comparación entre tipos de cámaras 3D [27].

Por último tenemos las cámaras de tiempo de vuelo, las cuales utilizan la velocidad de la luz como constante para calcular la profundidad, la mayoría de estas cámaras utilizan la luz láser y un sensor de esta misma luz para su funcionamiento [27].

Capítulo 3
Desarrollo del Proyecto

En este capítulo se presenta el diseño de un algoritmo y su implementación en un prototipo de sistema para calcular las dimensiones de objetos en imágenes digitales, el cual una vez terminado ofrecerá una herramienta que puedan utilizar los estudiantes para generar nuevo conocimiento u otras herramientas. A continuación se presenta el producto propuesto, la metodología empleada, así como el desarrollo de cada una de las fases.

3.1. Propuesta de Solución

Para resolver la problemática expresada en este documento, se propuso el desarrollo de un prototipo, el cual consiste en un algoritmo capaz de identificar y medir las dimensiones de los objetos para representar estos datos gráficamente.

3.2. Descripción de la metodología

El prototipo del sistema de medición de objetos propuesto en este proyecto de titulación fue desarrollado con un modelo basado en prototipo, descrito en [28]. Por su flexibilidad al momento del desarrollo y debido a los recursos que se quieren utilizar en el proyecto, no es necesario tener claros todos los requerimientos del sistema en un inicio, estos son un punto de

partida para generar un diseño rápido y generar un prototipo, esto hasta que se cuente con un sistema adecuado que no necesite modificaciones o adaptaciones. Al finalizar el proceso de mejora del prototipo, éste es implementado en el entorno donde funcionará finalmente. Este modelo está representado en la Figura 3.1 y cada fase se describe en las siguientes secciones.

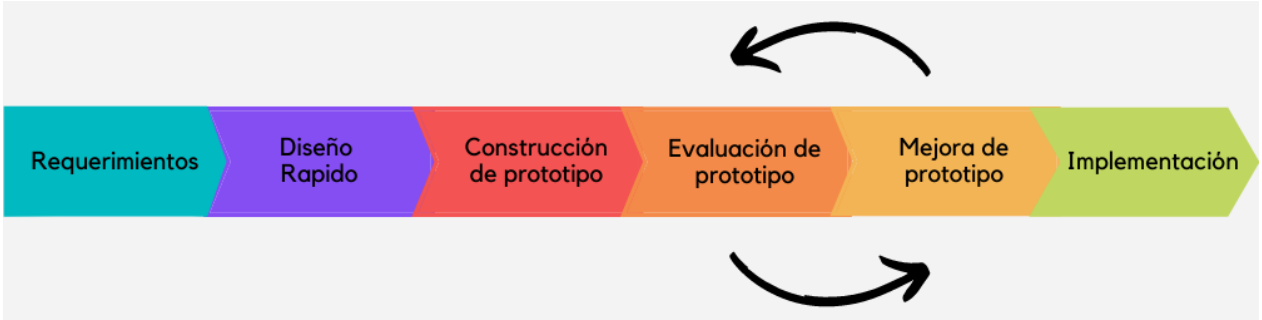


Figura 3.1: Metodología de prototipo utilizada para el sistema de medición.

3.3. Requerimientos

El propósito principal de este proyecto fue crear un software que sea capaz de integrarse con un sistema robótico utilizando los atributos de alto y largo de un objeto para funcionar. El reconocimiento del objeto tiene que estar dado por una red neuronal capaz de adaptarse a las capacidades de un modelo de alto o bajo rendimiento. Tomando esto en consideración, se determinaron los siguientes requisitos iniciales:

1. Un algoritmo que necesite de poco nivel de procesamiento y poca memoria para funcionar.
2. El producto debe de medir los objetos con una precisión mínima del 80 %.
3. Lenguaje de programación *Python*.

4. Utilizar una red neuronal pre-entrenada para el reconocimiento de objetos con alto porcentaje de precisión.

5. El sistema debe ser capaz de reconocer y medir al menos dos tipos de objetos, botellas y vasos.

6. Utilizar una biblioteca especializada para el manejo de la cámara *Realsense D-435* e integrar el funcionamiento con la red neuronal elegida.

3.4. Diseño Rápido

Para esta sección se cubren algunas de las necesidades planteadas en el modulo anterior, las cuales definen el punto de inicio del proyecto para proceder a la mejora del código. En concreto este primer diseño se compone de tres fases, las cuales se describen a continuación:

1. En la primera fase se importan las bibliotecas y se definen los nombres de los objetos que la red neuronal pre-entrenada tiene la capacidad de detectar. Para cada tipo de objeto se crea un color único que posteriormente se usó para marcarlo en la imagen. Posteriormente se define la resolución, imágenes por segundo y el formato de la imagen, tanto la de color como la imagen de profundidad de la cámara, una vez terminada la configuración se activa el dispositivo para que comience a capturar imágenes. Este proceso está descrito en la Figura 3.2.

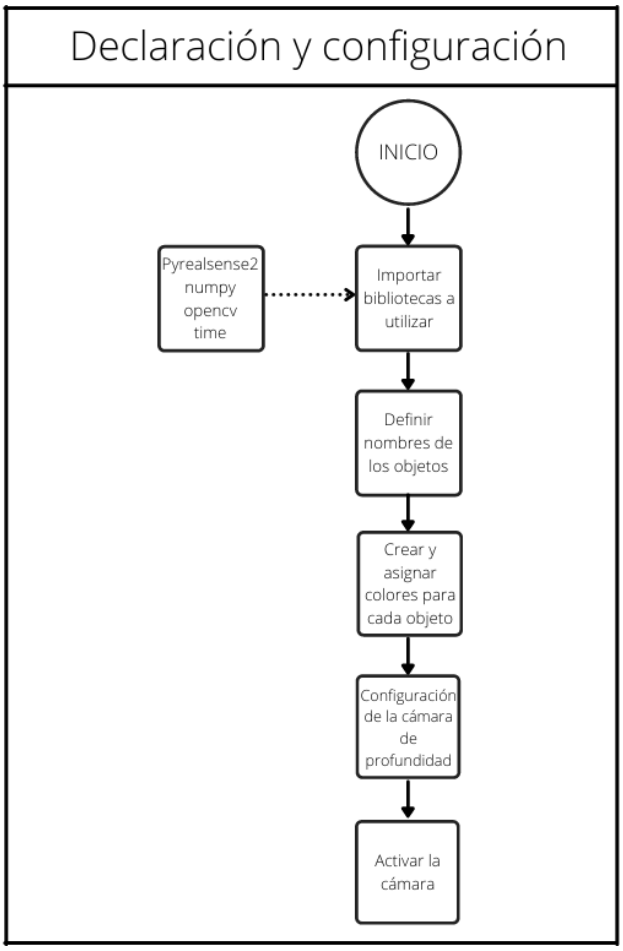


Figura 3.2: Declaración y configuración de la versión 1 del prototipo.

2. En la segunda fase se comienza a capturar el video, por cada imagen que toma la cámara ésta se procesa en la red neuronal para que reconozca los objetos en ella y marca su ubicación con el color correspondiente en una figura rectangular, su nombre o etiqueta y la distancia del centro de la figura. El programa repetirá este proceso de la segunda fase hasta que el usuario lo decida presionando una tecla específica 3.3.

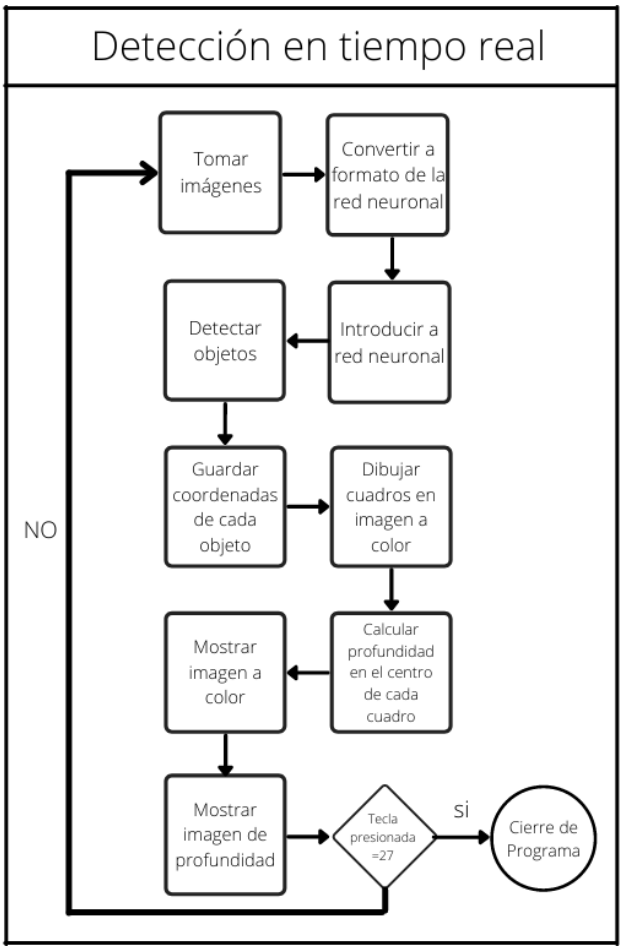


Figura 3.3: Fase de detección y cierre de la versión 1

3. Por último una vez que se presiona la tecla, el programa detiene la captura de video y cierra la aplicación.

3.5. Construcción de prototipo

Para cubrir los requerimientos propuestos se empieza por definir el lenguaje de programación a utilizar. Las características técnicas que más sobresalen de las redes neuronales son su aumento de complejidad entre más grandes sean y su alto consumo de recursos computacio-

nales debido a los miles de cálculos por segundo que se hacen. Tomando esto como referencia se optó por utilizar el lenguaje *Python* en su versión 3.7, si bien es cierto que ya existen versiones más recientes del mismo, esta versión cuenta con amplias bibliotecas actualizadas de inteligencia artificial para un funcionamiento más estable, cosa que en las nuevas versiones existen casos en los que no hay bibliotecas aun para ciertas tareas.

Tabla 3.1: Versión de Python y bibliotecas principales utilizadas en el prototipo

Biblioteca	**Versión**
python	3.7.11
opencv-python	4.5.3.56
pyrealsense2	2.49.0.3474
numpy	1.19.5

Se creó un ambiente virtual en *Anaconda Distribution*, este programa se utilizó por su diseño e interfaz gráfica, además de que muestra compatibilidad con la mayoría de bibliotecas necesarias para el desarrollo de programas en *Python*, fueron las razones de mayor peso para utilizarlo. Así se evitó que programas externos o versiones instaladas del lenguaje interfirieran con el funcionamiento del proyecto. Estas especificaciones de lenguaje de programación y las versiones de las principales bibliotecas utilizadas se muestran en la Tabla 3.1.

Los productos de la marca *Realsense* cuentan con su propia biblioteca tanto en *Python* como en lenguaje de programación C. En el caso de *Python* se utilizó *Pyrealsense2*, la cual se encarga de la configuración y funcionamiento de la cámara. La biblioteca OpenCV demuestra utilidad en este prototipo pues contiene métodos específicos para las redes neuronales tipo *Darknet* (YOLO). En conjunto con la biblioteca *numpy* para el manejo de las matrices y cálculos.

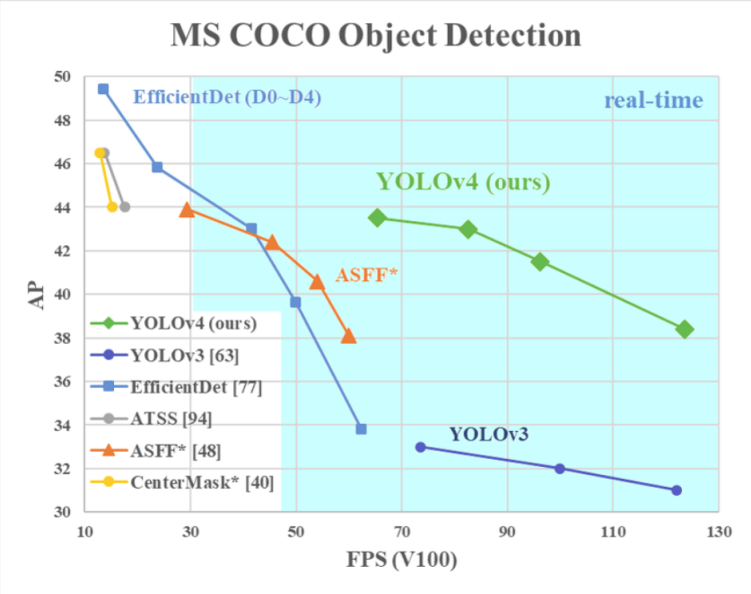


Figura 3.4: Rendimiento de la red neuronal YOLOv4 en tiempo real [24].

Para la parte de la red neuronal era necesaria una que cumpliera con un balance adecuado entre rapidez de detección y costo computacional. Tal es el caso de *YOLO* (You Only Look Once), pues su nombre y su diseño se tratan de que el modelo solo hace un barrido rápido a la imagen y a partir de ahí toma las detecciones con más probabilidad, de ahí su rapidez. En [24] se habla de las diferentes arquitecturas, componentes y pruebas que le hicieron a esta red, en las cuales mostraba un mejor rendimiento en comparación a otras en el campo de detección en tiempo real, obteniendo una precisión promedio o "AP"(*Average Precision* por sus siglas en inglés) cercanas a los 50 puntos a una velocidad mayor de 60 imagenes por segundo (FPS por sus siglas en inglés) lo que la hizo una candidata adecuada para el proyecto que se realizó en este documento como se muestra en la Figura 3.4.

A continuación se explica de forma detallada el código utilizado para el diseño inicial. El prototipo hace uso de tres librerías principales las cuales se describen a continuación:

1. Pyrealsense2: Es una adaptación de la biblioteca especializada de las cámaras *Intel*

Algoritmo 1: Fase de declaración y configuración

Datos: Biblioteca *Pyrealsense2* como *rs*
Biblioteca *numpy* como *np*
Biblioteca *OpenCV* como *cv2*
Archivo *yolov4.cfg*
Archivo *yolov4.weights*
Archivo *coco.names*

; /* Configuración y variables de red neuronal */

1 $objetos = Nombresaarreglo(coco.names)$;
2 $colores = asignarcolores(objetos)$;
3 $archivo\_red = ruta\_a(yolov4.cfg)$;
4 $pesos\_neuronas = ruta\_a(yolov4.weights)$;
5 $red = cv2.definir\_tipored(archivo\_red, pesos\_neuronas)$;
6 $red.Utilizar\_soloCPU()$;
7 $ln = red.Obtener\_capasalida()$;

; /* Configuración y variables de cámara */

8 $pipeline = rs.detectar\_camara()$;
9 $ancho = 640$;
10 $altura = 480$;
11 $config = rs.config\_camara()$;
12 $config.habilitar\_video(rs.video.profundidad, ancho, altura, rs.formato.z16, 30)$;
13 $config.habilitar\_video(rs.video.color, ancho, altura, rs.formato.bgr8, 30)$;
14 $perfil = pipeline.Encender(config)$;
15 $sensor\_profundidad = perfil.dispositivo().sensor\_profundidad()$;
16 $escala\_profundidad = sensor\_profundidad.escala()$;

Realsense. Su función principal es establecer comunicación con el aparato para su control y configuración. En el código se representa como *rs*.

2. Numpy: Es una biblioteca especializada en el cálculo numérico y el análisis de datos, utilizada mayormente para el manejo de gran volumen de datos. Es representada en el código como *np*.

3. Opencv: Biblioteca que contiene una amplia cantidad de algoritmos para la visión artificial, además de funciones auxiliares para la creación de ventanas con imágenes y modificación de las mismas de una manera sencilla. Es representada como *cv2* en el

código.

En el Algoritmo 1 se muestra la declaración de las variables iniciales necesarias para la configuración de la red neuronal (lineas 1-4), configuración de la cámara (lineas 8-16) y configuración del sistema para utilizar solo el CPU (linea 6). Esto compone la primera etapa descrita en la Figura 3.2.

Después del llamado de las bibliotecas se define la variable *objetos* la cual contiene todos los nombres que la red neuronal *yolov4* es capaz de reconocer por defecto en un arreglo. Por su parte la variable *colores* contiene un arreglo de números aleatorios del tamaño de *objetos*, esta variable se utiliza cuando es necesario pintar en la imagen los cuadros de detección, pues cada número representa un color.

Para la configuración de la red neuronal se usa la variable *red*, la cual utilizando la biblioteca de *OpenCV* llama al método especializado para lectura de arquitecturas *Darknet* en conjunto con el archivo de configuración *yolov4.cfg* y el archivo de pesos para las neuronas *yolov4.weights*. Este proceso hace que la variable *red* sea un atajo que se puede utilizar para el uso de métodos más específicos, como lo son el arrojar las neuronas no conectadas y el utilizar una configuración específica al momento de hacer los cálculos.

En la configuración de la cámara se define la variable *config* la cual mediante el método "config()" de la biblioteca *Pyrealsense2* define una resolución de las imágenes a color y profundidad en 640 píxeles de largo por 480 píxeles de alto a 30 FPS (Frames per second). Estos valores son especificados debido a que se busca un equilibrio entre rapidez de procesamiento y calidad de imagen, entre más resolución más tiempo será requerido para generar detecciones al momento de correr la imagen por la red.

Al final de esta fase se inicia la conexión entre la cámara y el sistema y se determina la escala del sensor del profundidad guardada en la variable *escala_profundidad*. Esta variable se utiliza cada vez que se quiera determinar la distancia de un píxel en una imagen de

profundidad a la cámara.

Algoritmo 2: Función para capturar imágenes

Datos: Variable global *pipeline*
Resultado: Imagen de profundidad $imagen\_profundidad$ tomada por cámara
Imagen a color $imagen\_color$ tomada por cámara

```
1 imagenes = pipeline.Capturar_imagenes();
2 imagen_profundidad = imagenes.Datos_profundidad;
3 imagen_color = imagenes.Datos_color;
4 imagen_profundidad = imagenaarreglo(imagen_profundidad);
5 imagen_color = imagenaarreglo(imagen_color);
```

Antes de comenzar a describir las siguientes fases es necesario comprender como el prototipo captura una imagen y la transforma a datos de profundidad. El método "capturar_imagenes()" es el que se encarga de esta función y esta representado en el Algoritmo 2. Esta función llama a la variable *pipeline* que tiene la configuración de la cámara previamente definida, invoca al método de captura de imagen, el cual nos arroja una imagen a color y una imagen a profundidad. Estos datos se almacenan en un objeto llamado *imagenes*, el cual se utiliza para guardar los datos de profundidad y de color en las variables $imagen\_profundidad$ e $imagen\_color$ respectivamente. Al final de este método los datos de cada imagen se transforman en arreglos numpy y son retornados, estos datos se insertan en

la red neuronal para generar las detecciones.

Algoritmo 3: Captura de video, detección y calculo de profundidad imagen

```
prueba:
    mientras Verdadero hacer
        img_profundidad, img_color = Capturar imagen de color y profundidad;
        blob = Preparar imagen a color para introducir a red neuronal;
        Introducir blob a red neuronal;
        salidas = Correr red neuronal;
        para salida En salidas hacer
            para deteccion En salida hacer
                confianza = Extraer puntaje máximo;
                si confianza >0.5 entonces
                    caja = Tomar coordenadas y medidas de deteccion;
                    Guardar caja en arreglo de cajas;
                    Guardar confianza en arreglo de confianzas;
                    Guardar nombre de objeto en arreglo de nombres;
                fin
            fin
        fin
        indices = Eliminar cajas no redundantes menores al 50 % de confiabilidad;
        si índices entonces
            para detección En índices hacer
                (x, y) = Extraer coordenadas de la detección;
                (w, h) = Extraer largo y altura de la detección;
                Dibujar en imagen a color la caja de detección;
                profundidad = Tomar dato de profundidad del centro;
                Colocar en imagen a color la profundidad;
            fin
        fin
    fin
Apagar cámara;
```

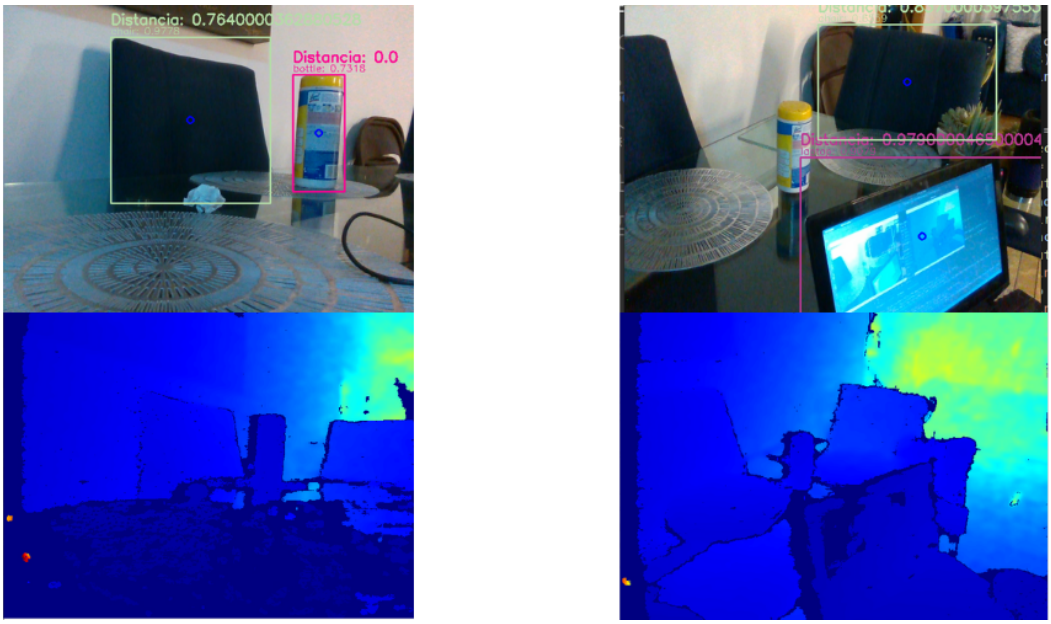
La última fase del programa descrita en el Algoritmo 3 está compuesta por un ciclo que se ejecutará hasta que el usuario presione la tecla con el codigo ASCII número 27 o *"ESC"*, dentro de una estructura *try*, la cual ayuda con los errores que puedan generarse durante todo el proceso de reconocimiento y medición. Se inicia el proceso llamando a la función *read_images()*, la cual nos devolverá la imagen de profundidad y la imagen a color, las cuales se guardan en dos variables. Se utiliza la imagen a color "img_color" para la creación de un *blob*. Estas estructuras son los datos que se le transferirán a la red neuronal para que procese la imagen. Se coloca la estructura dentro de la red neuronal y se inicia el proceso de detección.

Para guardar los datos que se recolectaron durante la detección se crean diferentes arreglos vacíos *cajas, confianzas, objetosIDs* al igual que variables que indiquen las dimensiones de las imágenes. Ahora se itera entre los resultados de la red neuronal "salidas" que contienen la información de todas las detecciones reconocidas en la imagen. Por cada "detección" se analiza si su confianza de detección "confianza" es mayor a 0.5 o dicho de otra manera 50 %. De ser esta condición cumplida se guardan sus coordenadas en la imagen y su tamaño en pixeles en la variable "caja" y esta misma se guarda en el arreglo "cajas", "confianza" se guarda en el arreglo "confianzas", y el nombre del objeto "classID" en el arreglo "objetosIDs".

Al finalizar este proceso de guardar todos los datos de los objetos reconocidos, se introducen en una función de la biblioteca *OpenCV* llamada "NMSBoxes", la cual aplica un algoritmo de filtrado a las detecciones realizadas para evitar duplicados en las mismas. El resultado de este código se guarda en la variable "indices" la cual contiene las coordenadas de cada detección "x, y" y su tamaño en pixeles que corresponde "w" al largo y "h" a la altura. Con estos datos se dibujan los rectángulos en "img_color" que representan el área de la imagen en la que YOLO detectó un objeto con la función "rectangle" de la librería *OpenCV*. Ademas de que haciendo uso de los datos de profundidad que nos proporciona "img_profundidad" se calcula la distancia de la cámara al pixel central del objeto. Esto da

por terminada la fase de detección y dibujado por lo que se muestra en pantalla el resultado. De no ser presionada la tecla "ESC", el código seguirá capturando imágenes y procesándolas para detección y dibujado. Cuando esta tecla es presionada el ciclo se detiene, la cámara se desactiva y el programa finaliza.

Debido al código explicado anteriormente se obtienen los resultados representados en la Figura 3.5, en la cual se puede observar el funcionamiento del programa.



(a) Prueba rápida 1.

(b) Prueba rápida 2.

Figura 3.5: Pruebas de la primera versión del prototipo.

3.6. Evaluación del prototipo

3.6.1. Primera iteración

El código inicial muestra un comportamiento inestable, al hacer pruebas y tener el programa corriendo por un tiempo mayor a los 30 segundos aproximadamente, comienza a fallar. Al igual que como se muestra en la Figura 3.6, al momento de que la red neuronal detecta un objeto en la imagen, en el cual el centro del mismo estuviera fuera de la misma, el programa se cierra. La red neuronal, por otro lado, hace su trabajo de la forma esperada, detecta objetos incluso cuando no se muestran en su totalidad.

```
Traceback (most recent call last):
  File "/home/rbs/Escritorio/CNN_mobjetos/Prototipo_1.py", line 89, in <module>
    profundidad = img_profundidad[int(x+(0.5*w)),int(y+(0.5*h))].astype("double")
IndexError: index 601 is out of bounds for axis 0 with size 480
```

Figura 3.6: Error en consola al detectar un objeto incompleto en pantalla.

Otro problema que se encuentra con la primera versión del prototipo es que la profundidad calculada es errónea debido a diferencias de dimensiones en la imagen a color, con la imagen de profundidad. Como se aprecia en la Figura 3.7, la caja de detección en la imagen de profundidad es del mismo tamaño que en la imagen a color, pero el objeto es mucho mas pequeño.

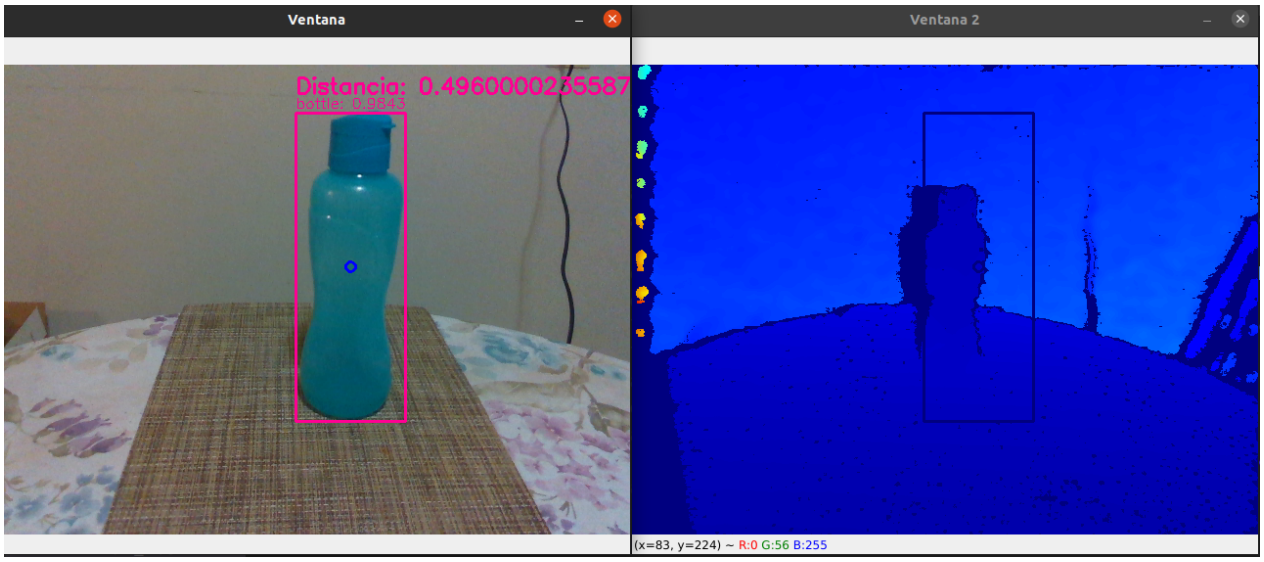


Figura 3.7: Discrepancia de magnitudes entre imagen a color e imagen de profundidad.

3.6.2. Segunda Iteración

El prototipo ahora se convierte en un algoritmo de calibración en el que como lo indica la Figura 3.8 primero se ingresan las medidas largo y altura del objeto en metros.

```
(mobjetos) rbs@rbs-Lenovo-Y50-70:~/Escritorio/CNN_mobjetos$
_mobjetos/Prototipo_v2.py
Largo del objeto? (metros): 0.058
Altura del objeto? (metros): 0.175
```

Figura 3.8: Ingreso de medidas reales del objeto en consola.

Después, procede la fase de Apuntado de la cámara y reconocimiento, en la que el usuario debe de acercar o alejar el objeto para que se encuentre en una de las seis medidas de calibración (0.5, 0.6, 0.7, 0.8, 0.9 ó 1 metro). Se puede apreciar que en la imagen de profundidad de la Figura 3.9 el filtro de "llenado de huecos" elimina en su mayoría los puntos de la imagen en los que su valor es 0. Esto se traduce a que se puede diferenciar con mayor facilidad los

objetos en ella.

Figura 3.9: Ejecución de fase de apuntado de cámara en la segunda versión del algoritmo.

Una vez que el objeto se encuentra a la distancia deseada se presiona cualquier tecla y se procede a extraer las coordenadas del objeto detectado y utilizarlas para copiar sus datos en un arreglo bidimensional que está representado en la Figura 3.10, que posteriormente se muestra en pantalla para que el usuario compruebe si el área que se extrae es la correcta.

Figura 3.10: Arreglo numpy de la región donde se detecta el objeto.

Después el programa muestra los valores por pixel calculados de la medida real del objeto entre el número de pixeles del área recortada. Se busca que las medidas por pixel, tanto el largo como la altura calculadas por el programa sean similares entre sí. Como se aprecia en la Figura 3.11, esta diferencia no es mayor a un milímetro por lo tanto son medidas aceptables y pueden ser consideradas cómo válidas para guardarse y utilizarse en la siguiente versión del prototipo.

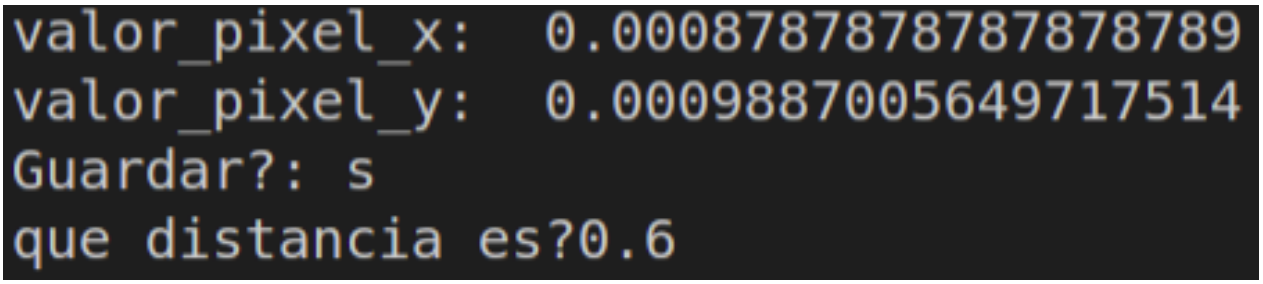
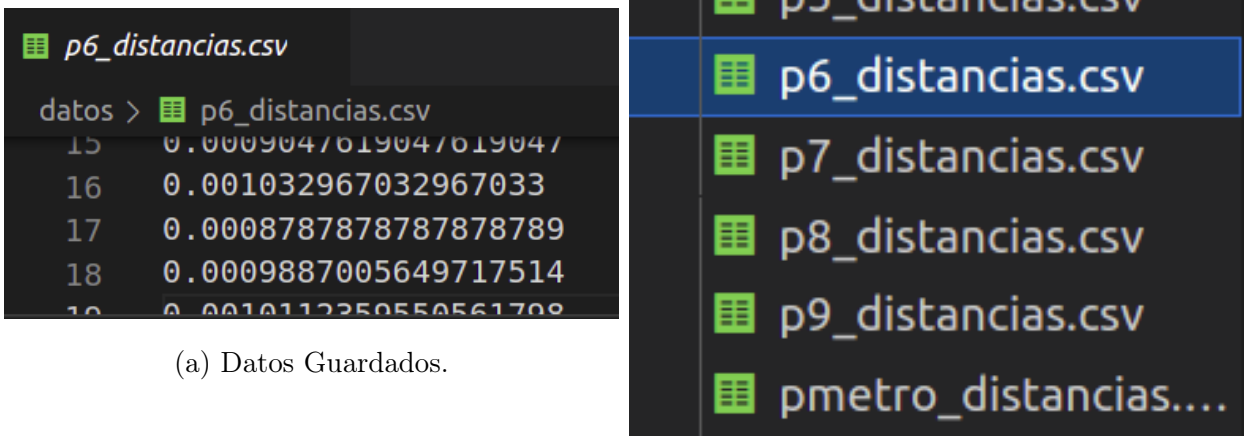


Figura 3.11: Resultados de los valores por pixel del largo y altura del objeto mostrados en consola.

En la Figura 3.12(a) se aprecia cómo están representados los datos de cada medición en los archivos creados por el programa y en la Figura 3.12(b) se encuentran los 6 archivos que el programa crea para guardar los datos de calibración dentro de la carpeta “datos”.



(a) Datos Guardados.

(b) Conjunto de archivos creados.

Figura 3.12: Calibración: Ejemplo de Archivos y valores por pixel.

El algoritmo funciona de la forma esperada, las tareas las realiza como debe, pues los problemas de detección de profundidad se han resuelto y el programa no genera errores si se detecta un objeto incompleto.

Con todas las mediciones hechas de 10 diferentes objetos se puede tener información suficiente para que el algoritmo pueda calcular las medidas de un objeto con una precisión igual o mayor a la buscada.

3.7. Mejora del prototipo

3.7.1. Primera iteración

El desempeño del programa inicial de acuerdo a la primera evaluación está dentro de un rango aceptable, ya que el propósito es cubrir los requerimientos simples de la sección 3.3.

El requerimiento de medición del objeto detectado es el que se explica cómo se desarrolla a continuación.

En primera instancia, hay que resolver el problema de cálculo de profundidad del centro en objetos que no se encuentran en la imagen en su totalidad, la solución empleada es la siguiente:

El problema radica en los valores cuando x ó y se vuelven negativos, esto se refiere a que el objeto se encuentra muy a la izquierda, muy arriba de la imagen o ambos. También cuando la suma de $x+w$ supera el tamaño del ancho de la ventana, cuando la suma de $y+h$ supera la altura de la misma o ambos. Con una condicionante que valide si uno de estos casos se cumple al momento en que la red neuronal detecta el objeto, al ser verdadera el programa debe omitir el dibujado de la caja de detección y el cálculo de la profundidad de su centro. Esta implementación puede visualizarse en el Algoritmo 4, el cual es una extracción de las líneas 21 a la 31 de el Algoritmo 3.

Algoritmo 4: Solución a problema de detección de profundidad

```
1  si índices entonces
2      para detección En índices hacer
3          (x, y) = Extraer coordenadas de la detección;
4          (w, h) = Extraer largo y altura de la detección;
5          si objeto fuera de imagen entonces
6              continuar;
7          fin
8          Dibujar en imagen a color la caja de detección;
9          profundidad = Tomar dato de profundidad del centro;
10         Colocar en imagen a color la profundidad;
11     fin
12 fin
```

Al resolver el problema anterior se continua con el problema de diferencia entre imágenes a color y profundidad. Esto se debe a que la cámara que toma las imágenes a color se encuentra en diferente lugar que la cámara de profundidad por lo que aunque estén configuradas con la misma resolución la imagen no es la misma y por lo tanto el objeto está ubicado en diferentes coordenadas en las dos imágenes, esto se soluciona de la siguiente manera:

Algoritmo 5: Solución a problema alineamiento de cámaras

1 $pipeline$ = Detectar la cámara de profundidad;
2 $ancho$ = 640 pixeles;
3 $altura$ = 480 pixeles;
4 $config$ = Aplicar configuración a la cámara;
5 Habilitar captura de imágenes de profundidad;
6 Habilitar captura de imágenes a color;
7 Encender cámara;
8 Declarar filtro de llenado de espacios;
9 Alinear imagen de profundidad con imagen a color;
10 Tomar escala de profundidad;

La biblioteca "Pyrealsense2" tiene un método diseñado para este caso, pues el comparar pixeles de una imagen a otra es una tarea regularmente usada. Hay que llamar al método como se puede visualizar en el Algoritmo 5 al igual que la invocación de los métodos de aplicado de filtros para la imagen de profundidad, estos últimos ayudan a la reducción de ruido y al llenado de huecos en las regiones sin datos. El alineamiento de imágenes y el aplicado de filtros está implementado en la función para capturar imágenes representada en el Algoritmo 6. Primero se espera a que el dispositivo capture las imágenes, después se aplican los diferentes filtros para al final alinear los resultados y al final transformarlos en arreglos numpy para su manipulación. Después de esta corrección, las imágenes están alineadas y el área de detección se encuentra en el lugar correcto en la imagen de profundidad, como puede verse en la Figura3.13

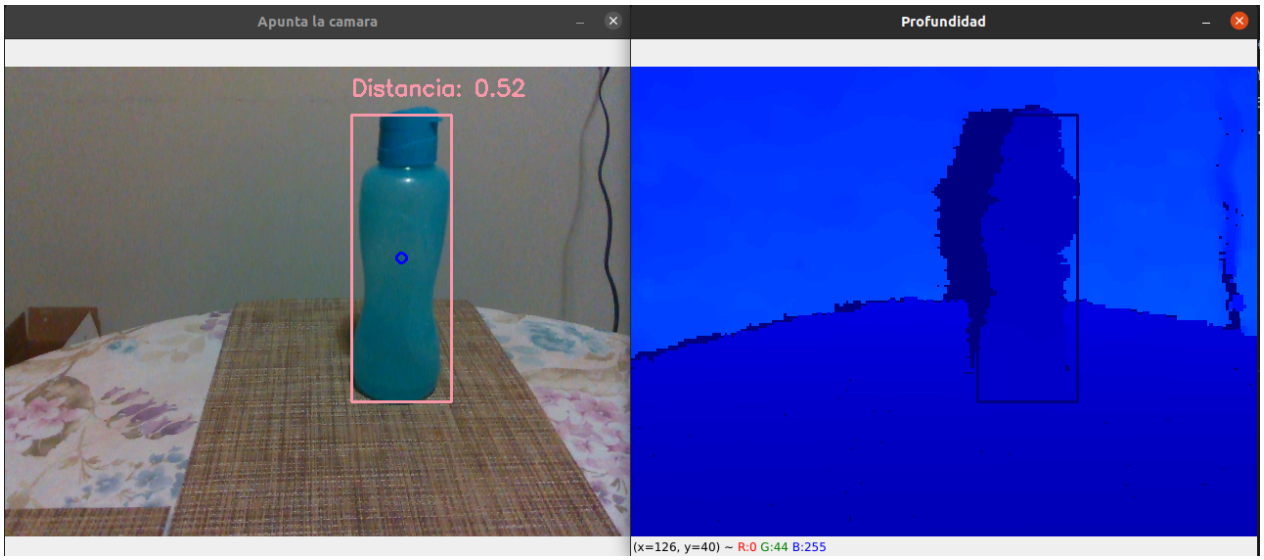


Figura 3.13: Comparación entre imagen a color e imagen de profundidad.

Algoritmo 6: Función para capturar imágenes

Datos: Variable global *pipeline*

Resultado: Imagen de profundidad $imagen\_profundidad$ tomada por cámara
Imagen a color $imagen\_color$ tomada por cámara
Arreglo de imagen de profundidad $profundidad\_arreglo$
Arreglo de imagen a color $color\_arreglo$

1 $imagenes = pipeline.Capturar\_imagenes();$
2 $filtrado = llenadoespacios.procesar(imagenes);$
3 $imagenes\_alineadas = alinear.procesar(filtrado);$
4 $imagen\_profundidad = imagenes\_alineadas.imagen\_profundidad;$
5 $imagen\_color = imagenes\_alineadas.imagen\_color;$
6 $profundidad\_arreglo = imagenaarreglo(imagen\_profundidad);$
7 $color\_arreglo = imagenaarreglo(imagen\_color);$

Con estas modificaciones el algoritmo ya es capaz de detectar un objeto y medir la distancia de la cámara al centro del objeto, ahora es necesario entender cómo el programa lo

medirá. Para fines de este proyecto se ha optado por hacer la medición a base de los pixeles de la región de la imagen. Esto puede lograrse si se encuentra la distancia aproximada de un pixel de la imagen a otro dependiendo la distancia a la que esté de la cámara. Para esta versión del algoritmo se elabora un método que crea un archivo *.csv*, calcula el valor por pixel (vpx) de un objeto cuyas medidas reales ya se conocen y se describe a continuación.

En el Algoritmo 3, en las líneas 24 y 25, se extraen las coordenadas de inicio de la caja de detección (x,y) y el tamaño en pixeles de largo y alto de la región (w,h) respectivamente. Lo que hace el Algoritmo 7 es tomar los valores de profundidad de esa región de la imagen, crea un arreglo numpy del tamaño (h,w) y coloca los datos en este arreglo nuevo llamado "profundidades".

Algoritmo 7: Recortar_region

Resultado: arreglo con datos de profundidad de region detectada $profundidades$

```
1  profundidades = np.arreglodeceros((h, w), float);
2  fila = 0;
3  columna = 0;
4  para altura En rango(y, y + h) hacer
5  |  columna = 0;
6  |  para largo En rango(x, x + w) hacer
7  |  |  profundidades[fila][columna] = img_profundidad.distancia(largo, altura);
8  |  |  columna+ = 1;
9  |  fin
10 |  fila+ = 1;
11 fin
```

Una vez que el arreglo de profundidades es elaborado se utilizan las medidas reales del objeto en x (largo) y en y (alto) y se dividen entre el número de pixeles w y h respectivamente. Esto resulta en dos valores que se guardan en archivos diferentes dependien-

do lo alejado que esté el objeto de la cámara, como se observa en el Algoritmo 8 Este algoritmo se enfoca en la medición de objetos a distancias no menores de 0.5 metros y no mayores a un metro por lo que con la finalidad de brindarle al programa un promedio de la distancia de pixel a pixel se midieron varios objetos de diferentes dimensiones para obtener una muestra confiable de posibles valores del mismo cada 10 centímetros. Estos datos se tomarón a partir del medio metro de distancia de la cámara, por lo que se cuentan con 6 archivos que contienen 20 resultados de valores por pixel cada uno.

Algoritmo 8: Función guardar_datos

```
1  Calcular valores por pixel de largo y altura;
2  Mostrar valores por pixel;
3  guardar = Entrada('Guardar? :');
4  si (guardar ==' S'orguardar ==' s') entonces
5  |  resp_dist = Entrada('que distancia es?');
6  |  si resp_dist ==' 0.5' entonces
7  |  |  archivo =' p5_distancias.csv';
8  |  si no, si resp_dist ==' 0.6' entonces
9  |  |  archivo =' p6_distancias.csv';
10 |  si no, si resp_dist ==' 0.7' entonces
11 |  |  archivo =' p7_distancias.csv';
12 |  si no, si resp_dist ==' 0.8' entonces
13 |  |  archivo =' p8_distancias.csv';
14 |  si no, si resp_dist ==' 0.9' entonces
15 |  |  archivo =' p9_distancias.csv';
16 |  si no, si resp_dist ==' 1' entonces
17 |  |  archivo =' pmetro_distancias.csv';
18 |  con abrir(archivo,' a') como a hacer;
19 |  Escribir valores en archivo correspondiente;
20 fin
```

3.7.2. Segunda iteración

Los datos para la calibración del algoritmo ya están completos y en esta mejora se utilizan para satisfacer el requerimiento faltante que es el cálculo de las dimensiones del objeto detectado. El algoritmo requiere que el usuario decida qué es lo que quiere hacer, calibrar el objeto para tener mas resultados de posibles valores por pixel ó medir un objeto. De elegir la opción de calibración el programa ejecutará la segunda versión del algoritmo. De lo contrario el programa toma todos los valores de los archivos y calcula el valor promedio respectivo a cada conjunto de archivos, ésta función esta descrita en la Figura 9.

Algoritmo 9: Sacar promedios

Datos: Biblioteca "Numpy" como *np*

Biblioteca interna "OS"

Resultado: arreglo ordenado $np.ordena(promedios)$

```
1 archivos = os.listadirectorios('datos/');
2 promedios = [];
3 para archivo En archivos hacer
4 |   datos = np.listararchivo('datos/' + archivo);
5 |   promedio = np.promedio(datos);
6 |   promedios.aadir(promedios);
7 fin
```

Al calcular los diferentes promedios el programa ahora solo necesita saber qué valor por pixel tomará, al momento de hacer el calculo del largo y la altura del objeto detectado. Por esta razón, la función "Valor_pixel" requiere la distancia existente entre la cámara y el objeto y el arreglo de promedios resultante de la función descrita anteriormente. En la Figura 10 se observa cómo toma el valor de *distancia* y lo compara con los diferentes rangos de distancias

para saber qué valor utilizar.

Algoritmo 10: Función Valor_pixel

Datos: Distancia en metros de la cámara al centro del objeto *distancia*

Arreglo $promedios_a$ de valores por pixel

Resultado: arreglo ordenado $np.ordena(promedios)$

```
si (distancia <= 1ydistancia > 9) entonces
    regresafloat(promedios[−1]);
si no, si distancia <= 0.9anddistancia > 0.8 entonces
    regresafloat(promedios[−2]);
si no, si distancia <= 0.8anddistancia > 0.7 entonces
    regresafloat(promedios[−3]);
si no, si distancia <= 0.7anddistancia > 0.6 entonces
    regresafloat(promedios[−4]);
si no, si distancia <= 0.6anddistancia > 0.5 entonces
    regresafloat(promedios[−5]);
si no, si distancia <= 0.5 entonces
    regresafloat(promedios[−6]);
en otro caso
    regresa−1;
```

Con estas mejoras el algoritmo presentado ya puede aproximar las medidas con porcentaje de precisión aceptable. En la Figura 3.14 se muestra un vaso de vidrio que mide 8.7 cm de largo y 15.5 cm de alto.

Figura 3.14: Resultado al ejecutar algoritmo para medición de objeto.

3.8. Implementación

El algoritmo resultante se describe en la Figura 3.15 al cual por motivos estéticos se implementaron funciones básicas irrelevantes para el funcionamiento, como el cálculo de medidas de la pantalla del equipo que se toman al inicio. Después, las instrucciones de configuración para la red neuronal se declaran para seguir con la configuración de la cámara. El programa requiere que el usuario elija si quiere calibrar más el programa o si desea medir un objeto con los datos que ya se tienen. Si se desea calibrar, el programa pide las medidas reales del objeto en metros, de haber elegido la opción de medición simplemente se comienza la fase de detección.

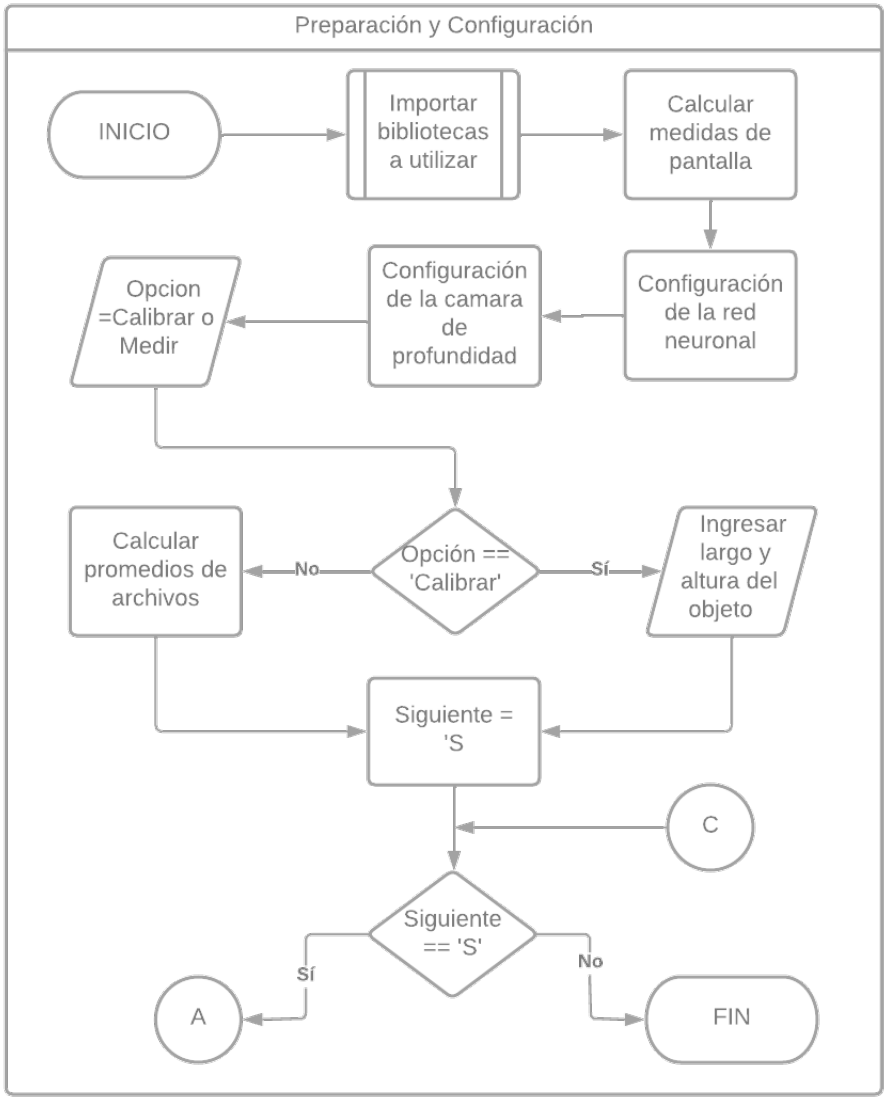


Figura 3.15: Fase de preparación y configuración.

En ambos casos se inicia la fase de detección, la cual enciende la cámara, toma la escala de profundidad del dispositivo y se inicia un ciclo que solo se detiene cuando se presiona la tecla “Enter”. Dentro de éste se toman imágenes de profundidad y a color la imagen de profundidad se utiliza para calcular distancias en las regiones en las que la red neuronal

detecte objetos de interés con la imagen a color. De existir objetos reconocidos se calcula su distancia. Dependiendo de la opción elegida al inicio del programa se calculan sus dimensiones o se toman sus datos para ingresarlos a los archivos de calibración.

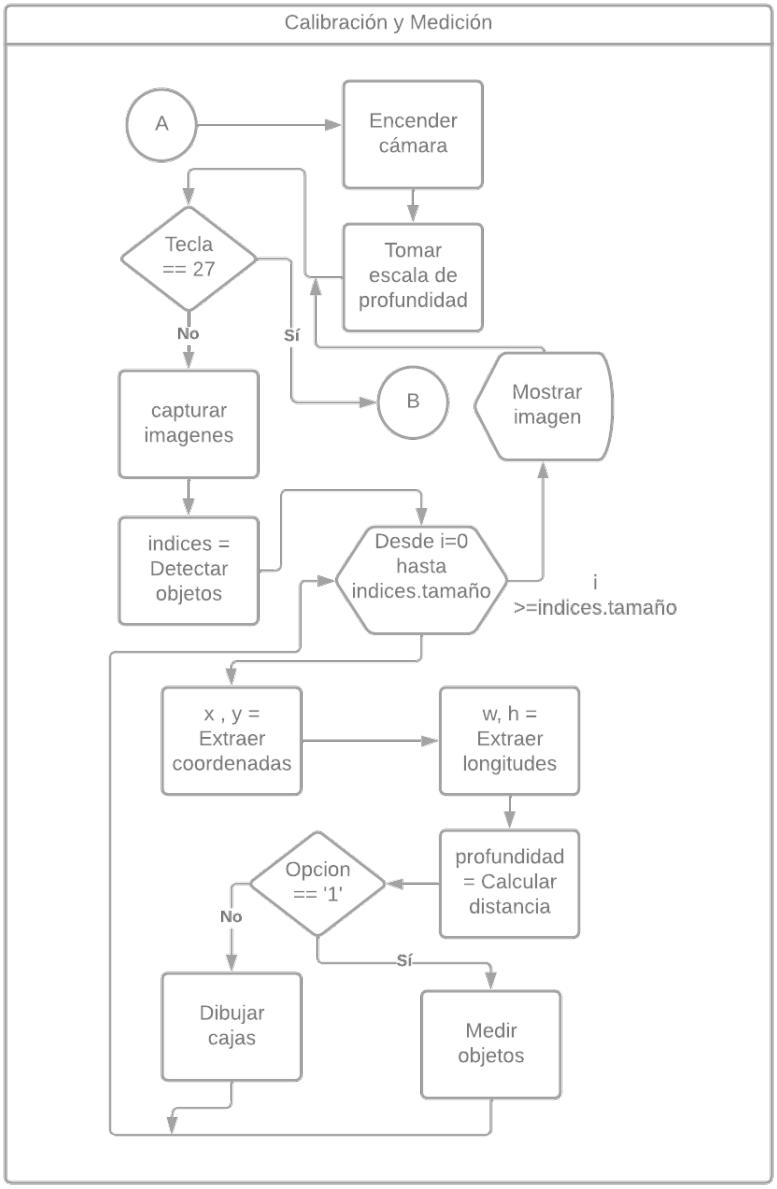


Figura 3.16: Fase de calibración y medición.

Cuando el usuario presiona la tecla “Enter” se evalúa si se eligió la opción de medición o la opción de calibración. De ser la primera el programa finaliza y de ser la segunda el programa pregunta si se quieren guardar los datos. De ser un valor “s” los datos se guardan en en el archivo que se indique dependiendo de la distancia del objeto. De ser un valor “n” no se hace nada y al finalizar se pregunta si el usuario quiere hacer otra medición y se guarda su respuesta para ser evaluada en la fase de calibración y medición.

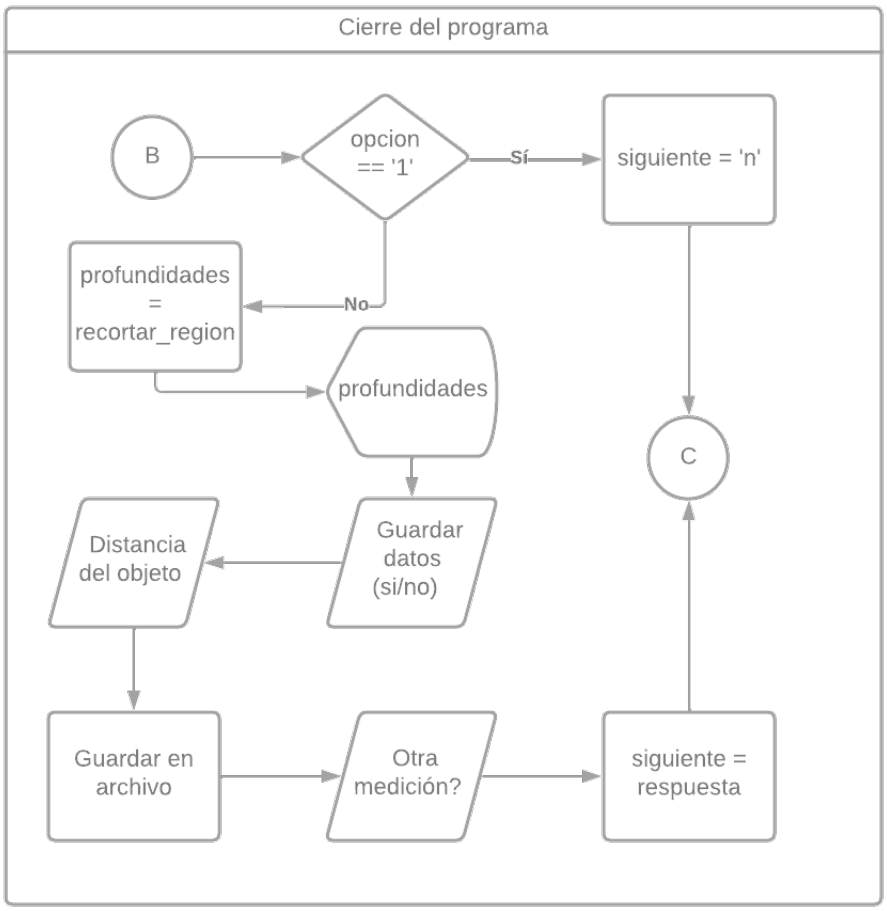


Figura 3.17: Fase de Cierre.

Capítulo 4
Resultados y Discusiones

En este apartado del documento se presentan los resultados de las pruebas de medición que se realizaron en un ambiente controlado con buena iluminación y un fondo blanco como se muestra en la Figura 4.1.

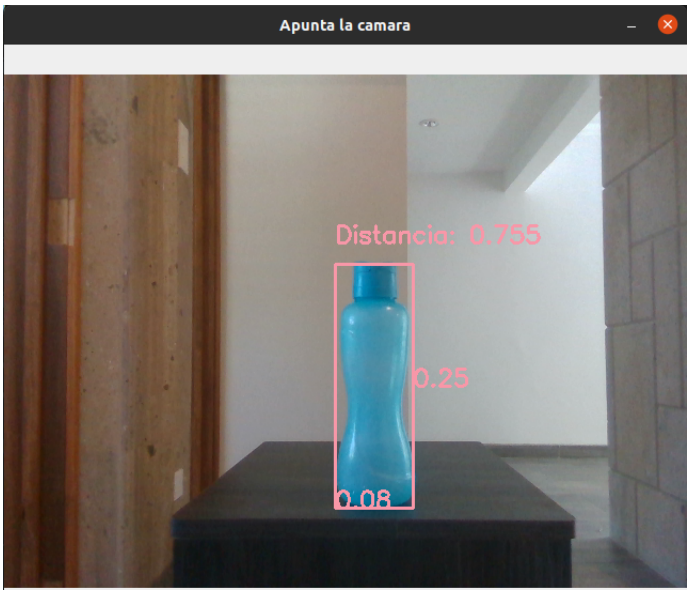


Figura 4.1: Ejemplo de ambiente de prueba con botella de tamaño mediano.

La detección de objetos está implementada con OpenCV utilizando su configuración predeterminada para modelos de redes neuronales de tipo “darknet”. También, se configuró el

programa para que no utilice recursos de hardware que no sean el procesador y memoria RAM, esto para que computadoras con pocos recursos puedan correrlo. A continuación se describe de una manera más detallada la información más relevante de las pruebas de medición realizadas.

Las características del equipo en el que se realizaron las pruebas son las siguientes:

- **Tipo de Equipo:** Notebook Lenovo Y50-70.
- **Sistema Operativo:** Ubuntu 20.04 LTS.
- **Procesador:** Intel(R) Core(TM) i7-4700HQ de 2.4 GHz.
- **Memoria RAM:** 16 GB.

La población que se tomó en cuenta para estas pruebas fue un conjunto de 10 objetos que se midieron cada 10 centímetros con la versión final del algoritmo a partir del medio metro de distancia de la cámara, esto dio como resultado una muestra de 20 mediciones por cada rango de distancia o un total de 120 mediciones realizadas las cuales se exponen a continuación.

El primer objeto fue un vaso de cristal cuyas medidas son 8.8 centímetros de largo y 16 centímetros de altura. Sus resultados pueden observarse en la Tabla 4.1, se puede apreciar un comportamiento muy regular en los datos a excepción de la medida en la altura del objeto cuando se encontraba a los 60 centímetros que obtuvo un error de casi 3 centímetros. Sin embargo el porcentaje del error respecto a la medida real del objeto se encuentra dentro del rango aceptable del 20 %.

Tabla 4.1: Resultados de medición objeto 1.

Vaso cristal			Largo Real	0.088 m	Altura Real	0.16 m
Distancia (m)	Largo Calculado (m)	Altura Calculada (m)	Error Largo (cm)	Error Altura (cm)	Error Largo (%)	Error Altura (%)
0.5	0.09	0.163	-0.2	-0.3	2.273	1.875
0.6	0.087	0.188	0.1	-2.8	1.136	17.5
0.7	0.085	0.168	0.30	-0.8	3.409	5
0.8	0.092	0.171	-0.4	-1.1	4.545	6.875
0.9	0.092	0.175	-0.4	-1.5	4.545	9.375
1	0.08	0.15	0.8	1	9.091	6.25

En los resultados del segundo objeto ubicados en la Tabla 4.2 se observan porcentajes bastante altos de error en la medición debido a que se observa una detección errónea, pues la red neuronal reconoce la parte inferior del objeto (botella), pero la parte superior (atomizador), cuyo largo es mayor al de la botella, siendo ésta una diferencia significativa.

Tabla 4.2: Resultados de medición objeto 2.

Botella aromatizante con atomizador			Largo Real	0.07 m	Altura Real	0.195 m
Distancia (m)	Largo Calculado (m)	Altura Calculada (m)	Error Largo (cm)	Error Altura (cm)	Error Largo (%)	Error Altura (%)
0.5	0.06	0.175	1	2	14.286	10.256
0.6	0.06	0.185	1	1	14.286	5.128
0.7	0.055	0.183	1.5	1.2	21.429	6.154
0.8	0.053	0.178	1.7	1.7	24.286	8.718
0.9	0.063	0.18	0.7	1.5	10	7.692
1	0.06	0.185	1	1	14.286	5.128

El tercer objeto cuyos resultados a la prueba están en la Tabla 4.3 fue elegido debido a su pequeña altura, de esta manera se pudo probar la capacidad de la red neuronal para reconocer objetos cuando se hagan más pequeños cuando se alejan de la cámara. El resultado

menos esperado fue en la fase de medición, pues se presentó un error significativo al calcular la altura cuando el objeto se encontraba a los 50 centímetros.

Tabla 4.3: Resultados de medición objeto 3.

Vaso cristal pequeño			Largo Real	0.088 m	Altura Real	0.073 m
Distancia (m)	Largo Calculado (m)	Altura Calculada (m)	Error Largo (cm)	Error Altura (cm)	Error Largo (%)	Error Altura (%)
0.5	0.083	0.089	0.5	-1.6	5.682	21.918
0.6	0.085	0.081	0.3	-0.8	3.409	10.959
0.7	0.076	0.071	1.2	0.2	13.636	2.74
0.8	0.084	0.079	0.4	-0.6	4.545	8.219
0.9	0.075	0.085	1.3	-1.2	14.773	16.438
1	0.079	0.077	0.9	-0.4	10.227	5.479

Después de ver el comportamiento del algoritmo con un objeto pequeño queda la incógnita de cómo respondería a un objeto más grande. Los datos representados en la Tabla 4.4 pertenecen al cuarto objeto y se registraron las primeras mediciones exactas a los 50 y 70 centímetros.

Tabla 4.4: Resultados de medición objeto 4.

Termo aluminio gris			Largo Real	0.082 m	Altura Real	0.31 m
Distancia (m)	Largo Calculado (m)	Altura Calculada (m)	Error Largo (cm)	Error Altura (cm)	Error Largo (%)	Error Altura (%)
0.5	0.082	0.316	0	-0.6	0	1.935
0.6	0.091	0.314	-0.9	-0.4	10.976	1.29
0.7	0.085	0.31	-0.3	0	3.659	0
0.8	0.084	0.301	-0.2	0.9	2.439	2.903
0.9	0.085	0.307	-0.3	0.3	3.659	0.968
1	0.088	0.3	-0.6	1	7.317	3.226

El objeto cinco, de acuerdo a los datos de la Tabla 4.5, tiene resultados prometedores, pues

no generó porcentajes de error mayores al 10 % en ninguno de los seis rangos de distancias al momento de realizar la prueba, incluso obtuvo dos mediciones exactas.

Tabla 4.5: Resultados de medición objeto 5.

Botella cloro			Largo Real	0.076 m	Altura Real	0.187 m
Distancia (m)	Largo Calculado (m)	Altura Calculada (m)	Error Largo (cm)	Error Altura (cm)	Error Largo (%)	Error Altura (%)
0.5	0.076	0.178	0	0.9	0	4.813
0.6	0.076	0.176	0	1.1	0	5.882
0.7	0.073	0.176	0.3	1.1	3.947	5.882
0.8	0.077	0.173	-0.1	1.4	1.316	7.487
0.9	0.07	0.174	0.6	1.3	7.895	6.952
1	0.078	0.18	-0.2	0.7	2.632	3.743

El sexto objeto presenta resultados muy cercanos a las medidas reales en la Tabla 4.6. Sin embargo, el objeto debía ser sujetado por una mano para que la red neuronal pudiera identificarlo como un celular y así lograr obtener sus medidas.

Tabla 4.6: Resultados de medición objeto 6.

Celular			Largo Real	0.078 m	Altura Real	0.16 m
Distancia (m)	Largo Calculado (m)	Altura Calculada (m)	Error Largo (cm)	Error Altura (cm)	Error Largo (%)	Error Altura (%)
0.5	0.08	0.175	-0.2	-1.5	2.564	9.375
0.6	0.08	0.17	-0.2	-1	2.564	6.25
0.7	0.075	0.16	0.3	0	3.846	0
0.8	0.082	0.162	-0.4	-0.2	5.128	1.25
0.9	0.075	0.16	0.3	0	3.846	0
1	0.065	0.157	1.3	0.3	16.667	1.875

El séptimo objeto se caracteriza por tener un largo menor a los demás pero una altura muy parecida, de acuerdo a la Tabla 4.7, donde se muestran sus resultados, el algoritmo

mostró ser capaz de llegar a aproximaciones de medidas aceptables incluso a una distancia alejada.

Tabla 4.7: Resultados de medición objeto 7.

Botella aromatizante			Largo Real	0.057 m	Altura Real	0.251 m
Distancia (m)	Largo Calculado (m)	Altura Calculada (m)	Error Largo (cm)	Error Altura (cm)	Error Largo (%)	Error Altura (%)
0.5	0.056	0.235	0.1	1.6	1.754	6.375
0.6	0.06	0.23	-0.3	2.1	5.263	8.367
0.7	0.058	0.235	-0.1	1.6	1.754	6.375
0.8	0.056	0.238	0.1	1.3	1.754	5.179
0.9	0.06	0.23	-0.3	2.1	5.263	8.367
1	0.055	0.247	0.2	0.4	3.509	1.594

Los resultados mostrados en la Tabla 4.8 muestran que las mediciones de la botella de cloro mediana tienen un mayor porcentaje de error cuando el objeto se encuentra a 80 centímetros de la cámara, aún así el algoritmo cumple el objetivo de no exceder el porcentaje de error de medición del 20 %.

Tabla 4.8: Resultados de medición objeto 8.

Botella cloro mediana			Largo Real	0.072 m	Altura Real	0.268 m
Distancia (m)	Largo Calculado (m)	Altura Calculada (m)	Error Largo (cm)	Error Altura (cm)	Error Largo (%)	Error Altura (%)
0.5	0.074	0.249	-0.2	1.9	2.778	7.09
0.6	0.075	0.253	-0.3	1.5	4.167	5.597
0.7	0.078	0.245	-0.6	2.3	8.333	8.582
0.8	0.082	0.245	-1	2.3	13.889	8.582
0.9	0.077	0.25	-0.5	1.8	6.944	6.716
1	0.081	0.252	-0.9	1.6	12.5	5.97

Los resultados de medición del algoritmo de medición en el objeto nueve plasmados en la

Tabla 4.9 muestran un resultado constante a pesar de estar en diferentes distancias.

Tabla 4.9: Resultados de medición objeto 9.

Botella jabón para platos			Largo Real	0.11 m	Altura Real	0.286 m
Distancia (m)	Largo Calculado (m)	Altura Calculada (m)	Error Largo (cm)	Error Altura (cm)	Error Largo (%)	Error Altura (%)
0.5	0.121	0.275	-1.1	1.1	10	3.846
0.6	0.115	0.256	-0.5	3	4.545	10.49
0.7	0.115	0.264	-0.5	2.2	4.545	7.692
0.8	0.115	0.26	-0.5	2.6	4.545	9.091
0.9	0.111	0.263	-0.1	2.3	0.909	8.042
1	0.115	0.263	-0.5	2.3	4.545	8.042

En los resultados pertenecientes al último objeto ubicados en la Tabla 4.10 puede observarse cómo el algoritmo aumenta su porcentaje de error en los rangos intermedios de medición de 70, 80 y 90 centímetros. No obstante se obtiene una medida acertada en la distancia mas lejana de medición.

Tabla 4.10: Resultados de medición objeto 10.

Botella vidrio cerveza			Largo Real	0.055 m	Altura Real	0.23 m
Distancia (m)	Largo Calculado (m)	Altura Calculada (m)	Error Largo (cm)	Error Altura (cm)	Error Largo (%)	Error Altura (%)
0.5	0.058	0.21	-0.3	2	5.455	8.696
0.6	0.059	0.21	-0.4	2	7.273	8.696
0.7	0.061	0.21	-0.6	2	10.909	8.696
0.8	0.063	0.215	-0.8	1.5	14.545	6.522
0.9	0.061	0.213	-0.6	1.7	10.909	7.391
1	0.055	0.21	0	2	0	8.696

4.1. Discusiones

De acuerdo a los resultados presentados en la sección anterior se pueden hacer las siguientes observaciones. La red neuronal utilizada para la detección es esencial, no solo para detectar un objeto en la imagen, sino para delimitar también dónde comienza y termina un objeto. Los porcentajes de error de la medida tomada por el algoritmo con respecto a la medida real del objeto es una representación de qué tan exacto es el algoritmo al momento de hacer las mediciones. Para hacer un mejor análisis se clasificaron los porcentajes de error en dos grupos, largos, presentados en la Figura 4.2 y alturas, presentados en la Figura 4.3. Se puede notar que para cada objeto representado con un numero en el eje x de las gráficas existen seis puntos que representan el error de medición a cierta distancia de la cámara.

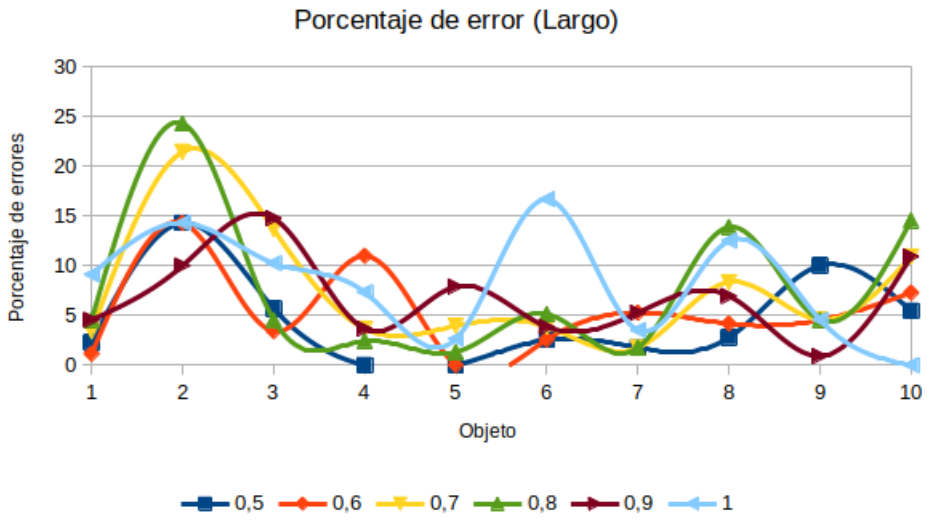


Figura 4.2: Gráfica de porcentaje de errores respecto a medidas de largos.

Los valores de la gráfica no varían mas del 20 % de error, incluso en algunos objetos las medidas fueron precisas. Sin embargo, existe al menos un punto a considerar en cada gráfica. En la Figura 4.2 existe un porcentaje que se acerca al 25 % y es el largo de una botella de aromatizante que tiene en la cabeza un atomizador, al momento de hacer las pruebas, la red

neuronal solo reconoce la botella pero no el complemento, el cual tiene una mayor longitud.

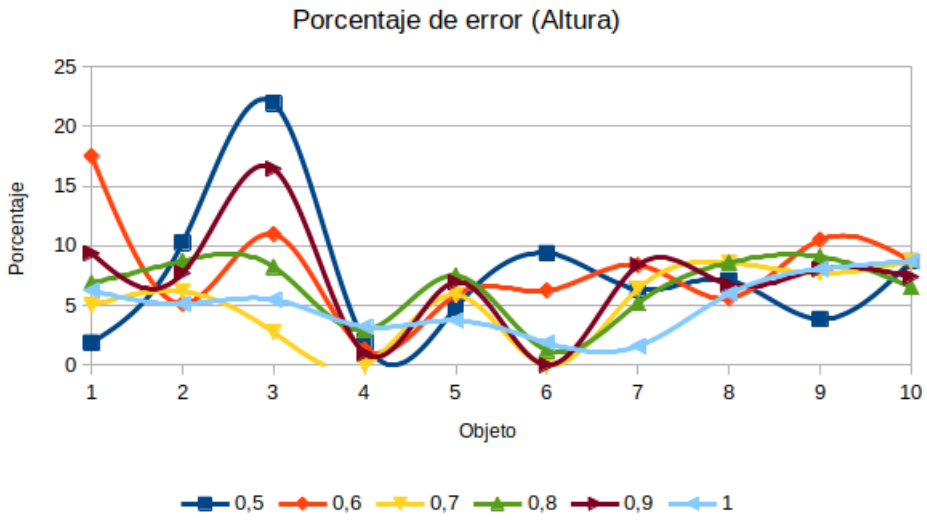


Figura 4.3: Gráfica de Porcentaje de errores respecto a medidas de alturas.

En la Figura 4.3 Los porcentajes más altos de error se dieron en un vaso de cristal con una altura de 7 centímetros. Si bien la diferencia se dio cuando el objeto se encontraba mas cerca de la cámara, el error en su medición puede deberse al material del objeto. Al sacar la media de los porcentajes de error de forma total y clasificándolos de acuerdo a la distancia del objeto a la cámara, como se observa en la Figura 4.4, se obtiene un error promedio mayor a los 80 y 90 centímetros que en la distancia más lejana, que es el metro, esto puede deberse a que hacen falta más valores de calibración a estas distancias para mejorar su efectividad. Sin embargo, de acuerdo a los resultados de estas pruebas, el porcentaje de efectividad de este algoritmo es mayor al 80 %.

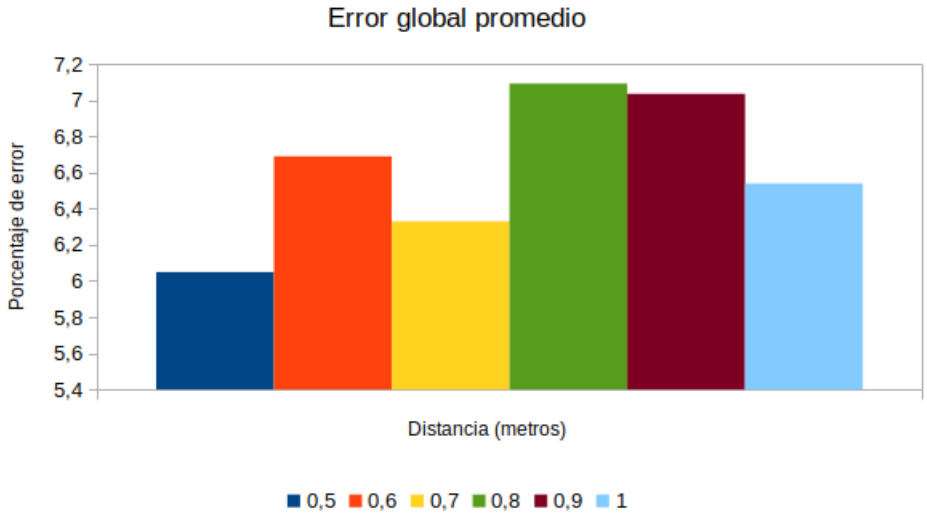


Figura 4.4: Media de porcentaje de error total de acuerdo a la distancia del objeto.

Capítulo 5
Conclusiones

El prototipo propuesto utiliza una sola cámara de profundidad y un equipo de cómputo desactualizado y sin hardware especializado. Sin embargo es posible obtener resultados de el aunque el equipo en el que se ejecuta cuente con poco poder de procesamiento. El sistema utiliza para la detección de objetos la red neuronal YOLO versión 4 que se especializa en reconocimiento en tiempo real. El algoritmo para la medición de los objetos es un método que no se había realizado dentro de la investigación hecha en diferentes documentos. Sin embargo debido a que el diseño de la red neuronal no es especializado en reconocimiento de contornos, el sistema no mide la altura y el largo de los objetos de una manera 100 % precisa, pues la medición del objeto depende de la región que la red neuronal detecte como tal, pero se consiguió un porcentaje de efectividad mayor al 80 % en las pruebas realizadas.

5.1. Con respecto al objetivo de la investigación

El prototipo de medición de objetos con redes neuronales hecho en el lenguaje de programación *python* es capaz de reconocer un objeto y calcular una aproximación de sus medidas (largo y altura).Las pruebas sugieren que el algoritmo cumple con un porcentaje de precisión en promedio del 93 % en un ambiente controlado midiendo solo un objeto. La red neuronal YOLOv4 es eficiente al detectar los objetos capturados por la cámara aunque se encuentren

a distancias alejadas sin embargo las cajas de detección que genera no son acertadas en su totalidad, lo que genera errores al momento de hacer las mediciones con el algoritmo.

5.2. Recomendaciones para futuras investigaciones

Como recomendaciones futuras el uso de diferentes estructuras de redes neuronales enfocadas a la segmentación podrían mostrar mejores resultados que las que solo se enfocan en detección de objetos. La cámara $D - 435$ *Realsense* de Intel, mostró un desempeño por encima de lo deseado al momento de su uso, la precisión de sus sensores es de alta calidad y su biblioteca especializada de uso libre la hace una herramienta recomendable para futuros proyectos enfocados al análisis de imagenes. Mejorar la precisión del valor por pixel ayudaría a la efectividad del algoritmo cuando se encuentra a distancias localizadas entre los rangos de 10 centímetros.

Bibliografía

[1] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, 2018.

[2] A. Alcides, D. Cantero, and J. E. Alcides, "Visión por computadora: identificación, clasificación y seguimiento de objetos.," *FPUNE Scientific*, no. 10, 2016.

[3] E. Marcos Viforcos, "Aplicación de las cámaras 3d al reconocimiento de actividades," B.S. thesis, 2012.

[4] M. J. Abásolo, A. Mitaritonna, S. Castañeda, *et al.*, "Aplicaciones de visión por computador, realidad aumentada y tvdi," in *Workshop de Investigadores en Ciencias de la Computación*, vol. 20, 2018.

[5] E. Cippitelli, F. Fioranelli, E. Gambi, and S. Spinsante, "Radar and rgb-depth sensors for fall detection: A review," *IEEE Sensors Journal*, vol. 17, no. 12, pp. 3585–3604, 2017.

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

[7] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[8] L. Keselman, J. Iselin Woodfill, A. Grunnet-Jepsen, and A. Bhowmik, "Intel realsense stereoscopic depth cameras," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1–10, 2017.

[9] S. Zia, B. Yuksel, D. Yuret, and Y. Yemez, "Rgb-d object recognition using deep convolutional neural networks," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 896–903, 2017.

[10] L. García Tena, H. Sossa, A. Alvarado-Iniesta, *et al.*, "Reconocimiento de objetos en una plataforma robótica móvil.," *CULCyT: Cultura Científica y Tecnológica*, vol. 12, no. 55, pp. 70–82, 2015.

[11] D. B. Monge, "Reconocimiento de objetos en 3d. utilizando sensores de visión y profundidad de bajo coste," *Escuela de Ingeniería y Arquitectura. Universidad de Zaragoza*, 2012.

[12] L. Liu, W. Ouyang, X. Wang, *et al.*, "Deep learning for generic object detection: A survey," *International journal of computer vision*, vol. 128, no. 2, pp. 261–318, 2020.

[13] E. R. Caballero Barriga *et al.*, "Aplicación práctica de la visión artificial para el reconocimiento de rostros en una imagen, utilizando redes neuronales y algoritmos de reconocimiento de objetos de la biblioteca opencv," 2017.

[14] J. Estarita, A. Jim, J. Brochero, *et al.*, "Sistema de reconocimiento de objetos en tiempo real," *Investigación y desarrollo en TIC*, vol. 8, no. 2, pp. 41–45, 2017.

[15] C. Montoya Holguin, J. A. Cortés Osorio, and J. A. Chaves Osorio, "Sistema automático de reconocimiento de frutas basado en visión por computador," *Ingeniare. Revista chilena de ingeniería*, vol. 22, no. 4, pp. 504–516, 2014.

[16] G. J. M. Hernández Granados, "Reconocimiento de objetos removidos de una escena, aplicando técnicas de visión por computadora.," 2016.

[17] J. V. Rebaza, "Detección de bordes mediante el algoritmo de canny," *Escuela Académico Profesional di Informática. Universidad Nacional de Trujillo*, vol. 4, 2007.

[18] G. Litjens, T. Kooi, B. E. Bejnordi, *et al.*, "A survey on deep learning in medical image analysis," *Medical image analysis*, vol. 42, pp. 60–88, 2017.

[19] Z. Zou, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *arXiv preprint arXiv:1905.05055*, 2019.

[20] S. Lawrence, C. L. Giles, A. C. Tsoi, *et al.*, "Face recognition: A convolutional neural-network approach," *IEEE transactions on neural networks*, vol. 8, no. 1, pp. 98–113, 1997.

[21] W. Zhiqiang and L. Jun, "A review of object detection based on convolutional neural network," in *2017 36th Chinese Control Conference (CCC)*, pp. 11104–11109, IEEE, 2017.

[22] E. Sepulveda Valdivia, "Redes neuronales convolucionales, reconocimiento de imágenes y pronósticos financieros," Master's thesis, Universitat Politècnica de Catalunya, 2019.

[23] J. Durán Suárez, *Redes neuronales convolucionales en R: Reconocimiento de caracteres escritos a mano.* PhD thesis, Universidad de sevilla, 2017.

[24] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020.

[25] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, pp. 740–755, Springer, 2014.

[26] C. Català Sarmiento, *Edición estereoscópica.* PhD thesis, 2014.

[27] Intel-Corporation, "Beginner's guide to depth." [Internet]. Disponible en: `https://www.intelrealsense.com/beginners-guide-to-depth/`, 2019. Ultimo acceso 16 mayo 2022.

[28] D. Gutierrez, "Métodos de desarrollo de software," *Caracas: Universidad de los Andes*, 2011.

Apéndice A
Código Fuente

A.1. calibrar.py

```
1  import csv
2
3  def guardar_datos(medida_X, medida_y,w,h):
4    valor_pixel_x = float(medida_X) / w
5    valor_pixel_y = float(medida_y) / h
6
7    print("valor_pixel_x: ", valor_pixel_x)
8    print("valor_pixel_y: ", valor_pixel_y)
9    guardar = input('Guardar?: ')
10   if(guardar=='S' or guardar=='s'):
11     resp_dist = input('que distancia es?')
12     if resp_dist == '0.5':
13         archivo = 'p5_distancias.csv'
14     elif(resp_dist == '0.6'):
15         archivo = 'p6_distancias.csv'
16     elif(resp_dist == '0.7'):
17         archivo = 'p7_distancias.csv'
18     elif(resp_dist == '0.8'):
19         archivo = 'p8_distancias.csv'
20     elif(resp_dist == '0.9'):
```

```
21          archivo = 'p9_distancias.csv'
22      elif(resp_dist == '1'):
23          archivo = 'pmetro_distancias.csv'
24      with open("datos/"+archivo,'a',newline='') as f:
25          escritor = csv.writer(f)
26          escritor.writerow([valor_pixel_x])
27          escritor.writerow([valor_pixel_y])
```

Listado A.1: Codigo fuente de archivo que guarda los valores por pixel en archivos calibrar.

A.2. recorte_reg.py

```
1  import numpy as np
2  def recortar_region(x,y,w,h,img_profundidad):
3    profundidades = np.zeros((h,w),dtype=float)
4    fila=0
5    columna=0
6    for altura in range(y,y+h):
7      columna=0
8      for largo in range(x,x+w):
9          profundidades[fila][columna]=img_profundidad.get_distance(largo,
     altura)
10         columna+=1
11     fila+=1
12   return profundidades
```

Listado A.2: Código fuente de archivo que recorta la region detectada.

A.3. tam_pant.py

```
1  import subprocess
2
3  def tamano_pantalla():
4      size = (None, None)
5      args = ["xrandr", "-q", "-d", ":0"]
6      proc = subprocess.Popen(args,stdout=subprocess.PIPE)
7      for line in proc.stdout:
8          if isinstance(line, bytes):
9              line = line.decode("utf-8")
10             if "Screen" in line:
11                 size = (int(line.split()[7]),  int(line.split()[9][:-1]))
12     return size
```

Listado A.3: Codigo fuente de archivo que toma dimensiones de pantalla.

A.4. valor_pixel.py

```
1  import numpy as np
2  import os
3
4  def promedios():
5      archivos = os.listdir('datos/')
6      promedios = []
7      for archivo in archivos:
8        datos = np.genfromtxt('datos/'+archivo,dtype=float,delimiter=',')
9        promedio = np.mean(datos)
10       promedios.append(promedio)
11     return np.sort(promedios)
12
13 def valor_pixel(distancia, promedios_a):
14   if distancia >= 1:
15     return float(promedios_a[-1])
16   elif distancia >= 0.9 and distancia < 1:
17     return float(promedios_a[-2])
18   elif distancia >= 0.8 and distancia < 0.9:
19     return float(promedios_a[-3])
20   elif distancia >= 0.7 and distancia < 0.8:
21     return float(promedios_a[-4])
22   elif distancia >= 0.6 and distancia < 0.7:
23     return float(promedios_a[-5])
24   elif distancia >= 0.4 and distancia < 0.6:
25     return float(promedios_a[-6])
26   else :
27     return -1
```

Listado A.4: Codigo fuente de archivo que regresa el valor por pixel.

A.5. video.py

```
1  import cv2
2  import numpy as np
3
4  def detectar_objetos(red, capa_salida ,arreglo_color):
5
6    blob = cv2.dnn.blobFromImage(arreglo_color, 1/255.0, (640,480), swapRB=
       True, crop=False)
7    r = blob[0,0,:,:]
8
9    red.setInput(blob)
10   salidas = red.forward(capa_salida)
11
12   cajas = []
13   confianzas = []
14   objetosIDs = []
15   al , an = arreglo_color.shape[:2]
16
17   for salida in salidas:
18       for deteccion in salida:
19           puntajes = deteccion[5:]
20           clasesID = np.argmax(puntajes)
21           confianza = puntajes[clasesID]
22           if confianza > 0.5:
23               caja = deteccion[:4] * np.array([an, al, an, al])
24               (centroX, centroY, ancho, alto) = caja.astype("int")
25               x = int(centroX - (ancho / 2))
26               y = int(centroY - (alto / 2))
27               caja = [x, y, int(ancho), int(alto)]
28               cajas.append(caja)
29               confianzas.append(float(confianza))
30               objetosIDs.append(clasesID)
```

```
31
32    indices = cv2.dnn.NMSBoxes(cajas, confianzas, 0.5, 0.4)
33    return indices, cajas, objetosIDs , al, an
```

Listado A.5: Codigo fuente de archivo que detecta objetos en imagenes.

A.6. Prototipo_v3.py

```
import pyrealsense2 as rs
import numpy as np
import cv2
import valor_pixel as vpx
import tam_pant as tpantalla
import calibrar
import recorte_reg as region
import video

def crear_ventana(Nombre_ventana, img, x=0, y=0):
    cv2.namedWindow(Nombre_ventana)
    cv2.moveWindow(Nombre_ventana,x,y)
    cv2.imshow(Nombre_ventana, img)

def capturar_imagenes():
    # Espera de imagenes a color y de profundidad
    imagenes = pipeline.wait_for_frames()
    filtrado = llenadoespacios.process(imagenes).as_frameset()
    imagenes_alineadas = alinear.process(filtrado)
    imagen_profundidad = imagenes_alineadas.get_depth_frame()
    imagen_color = imagenes_alineadas.get_color_frame()
    # Convertir imagenes a arreglos numpy
    profundidad_arreglo = np.asanyarray(imagen_profundidad.get_data())
    color_arreglo = np.asanyarray(imagen_color.get_data())
    return  profundidad_arreglo, color_arreglo, imagen_profundidad,
   imagen_color

#largo y ancho pantalla
x_pantalla, y_pantalla = tpantalla.tamano_pantalla()

#Definir los nombres de los objetos y generar colores aleatorios
```

```
31 objetos = open('darknet/data/coco.names').read().strip().split('\n')
32 np.random.seed(42)
33 colores = np.random.randint(0, 255,size=(len(objetos),3),dtype='uint8')
34
35 # Configurar la red neuronal
36 archivo_red = "darknet/cfg/yolov4.cfg"
37 pesos_neuronas = "darknet/yolov4.weights"
38 red = cv2.dnn.readNetFromDarknet(archivo_red, pesos_neuronas)
39 red.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
40
41 #determinamos la capa de salida
42 ln = red.getLayerNames()
43 ln = [ln[i[0]-1] for i in red.getUnconnectedOutLayers()]
44
45 # Creamos la comunicacion entre el programa y la camara
46 pipeline = rs.pipeline()
47
48 #Configuracion de la camara
49 ancho=640
50 altura=480
51 config = rs.config()
52
53 pipeline_wrapper = rs.pipeline_wrapper(pipeline)
54 perfil = config.resolve(pipeline_wrapper)
55 dispositivo = perfil.get_device()
56
57 config.enable_stream(rs.stream.depth, ancho, altura, rs.format.z16, 30)
58 config.enable_stream(rs.stream.color, ancho, altura, rs.format.bgr8, 30)
59
60 llenadoespacios = rs.hole_filling_filter()
61 alinear_a=rs.stream.color
62 alinear = rs.align(alinear_a)
```

```
63
64 #Pantalla equipo
65 siguiente = 'S'
66
67 print('1.-  Medir')
68 print('2.-  Calibrar')
69 opcion = input('Elegir Opcion: ')
70 if(opcion == '2'):
71     medida_objeto_x=input('Largo del Objeto (metros): ')
72     medida_objeto_y=input('Altura del Objeto (metros)')
73 elif (opcion == '1'):
74     promedios = vpx.promedios()
75
76 #Transmision en tiempo real para direccion de c mara
77 while(siguiente == 'S' or siguiente == 's'):
78     #Activacion de comunicacion a la camara
79     perfil = pipeline.start(config)
80     #Obtener la escala del sensor de profundidad
81     sensor_profundidad = perfil.get_device().first_depth_sensor()
82     escala_profundidad = sensor_profundidad.get_depth_scale()
83
84     while(True):
85     #construimos un blob de la imagen tomada por la camara
86         arreglo_profundidad, arreglo_color, img_profundidad, img_color =
    capturar_imagenes()
87         indices, cajas, objetosIDs, al, an = video.detectar_objetos(red,ln
    , arreglo_color)
88
89         if len(indices) > 0:
90             for i in indices.flatten():
91                 (x, y) = (cajas[i][0], cajas[i][1])
92                 (w, h) = (cajas[i][2], cajas[i][3])
```

```
93
94                  if(x<0 or y<0 or x+w>480 or y+h>640):
95                      print('Objeto fuera de RANGO')
96                      continue
97                  color = [int(c) for c in colores[objetosIDs[i]]]
98                  profundidad = round(img_profundidad.get_distance(int(x
    +(0.5*w)),int(y+(0.5*h))),3)
99                  cv2.rectangle(arreglo_color, (x, y), (x + w, y + h), color
    , 2)
100                 if(opcion == '1'):
101                     valor_vpx = vpx.valor_pixel(profundidad, promedios)
102                     if(valor_vpx==-1):
103                         continue
104                     objeto_largo = w * valor_vpx
105                     objeto_alto = h * valor_vpx
106                     cv2.putText(arreglo_color, "Distancia: "+str(round(
    profundidad,3)), (int(x),int(y-20)),cv2.FONT_HERSHEY_SIMPLEX,0.75,color
    , 2)
107                     cv2.putText(arreglo_color, str(round(objeto_alto,3)),
    (int(x+w),int(y+(h/2))),cv2.FONT_HERSHEY_SIMPLEX,0.75,color, 2)
108                     cv2.putText(arreglo_color, str(round(objeto_largo,3)),
     (int(x),int(y+h)),cv2.FONT_HERSHEY_SIMPLEX,0.75,color, 2)
109                 elif opcion == '2':
110                     cv2.circle(arreglo_color,(int(x+(0.5*w)),int(y+(0.5*h)
    )),5,(255, 0, 0), 2)
111                     cv2.putText(arreglo_color, "Distancia: "+str(round(
    profundidad,3)), (int(x),int(y-20)),cv2.FONT_HERSHEY_SIMPLEX,0.75,color
    , 2)
112         crear_ventana('Apunta la camara',arreglo_color,int((x_pantalla/2)
    -(an/2)),int((y_pantalla/2)-(al/2)))
113         j = cv2.waitKey(1)
114         if j == 13:
```

```
115             cv2.destroyWindow('Apunta la camara')
116             pipeline.stop()
117             break
118     if(opcion == '1'):
119         siguiente = 'n'
120         break
121     elif opcion == '2':
122         #Proceso de deteccion
123             #Crear arreglo de profundidades con coordenadas de detecci n
124         profundidades = region.recortar_region(x,y,w,h,img_profundidad)
125         #resultados de valores por pixel a diferentes distancias
126 
127         crear_ventana('Profundidades',profundidades,int((x_pantalla/4)-(an
    /2)),int((y_pantalla/2)-(al/2)))
128         cv2.waitKey(0)
129         cv2.destroyAllWindows()
130 
131         calibrar.guardar_datos(medida_objeto_x,medida_objeto_y,w,h)
132         siguiente = input("Otra medicion?")
```

Listado A.6: Codigo fuente prototipo final.