





Reconocimiento y estimación de distancia relativa de objetos en entornos controlados Recognition and estimation of relative distance of objects in controlled environments

E. Clemente-Rosas ^a, E. Luna-Taylor ^{a,*}, J. L. Gómez-Torres ^a, I. Villa-Medina ^a

^aTecnológico Nacional de México Campus La Paz, Boulevard Forjadores de Baja California Sur No.4720 C.P. 23080

Resumen

En este proyecto, se presenta un sistema de reconocimiento y clasificación de objetos que se encuentran en su entorno, para un robot de asistencia, así como la estimación de la distancia relativa de estos con respecto al robot. Para el reconocimiento y clasificación de los objetos, se aplicaron técnicas de visión artificial basadas en herramientas de segmentación semántica, como son redes neuronales convolucionales. Para la estimación de la distancia relativa de los objetos, una vez identificados, se implementó una técnica de visión estereoscópica. Los resultados de los experimentos muestran un 90.6 % de precisión en el reconocimiento y clasificación, y un error medio de 4.6 cm al estimar la distancia de los objetos.

Palabras Clave: Visión artificial, Redes neuronales convolucionales, Segmentación semántica, Visión estereoscópica, Inteligencia artificial.

Abstract

This project presents a system for recognition and classification of objects that are in its environment, for an assistance robot, as well as the estimation of their relative distance with respect to the robot. For the recognition and classification of objects, we apply artificial vision techniques based on semantic segmentation tools, such as convolutional neural networks. For the estimation of the relative distance of the objects, once identified, a stereoscopic vision technique was implemented. The results of the experiments show a 90.6 % accuracy in recognition and classification, and an average error of 4.6 cm when estimating the distance of the objects.

Keywords: Computer vision, Convolutional neural networks, Semantic segmentation, Stereoscopic vision, Artificial intelligence.

1. Introducción

Debido a la escasez y costos de personal de salud y de asistencia en muchos países, los pacientes que están postrados en cama debido a problemas de salud, accidentes o por envejecimiento, en ocasiones no cuentan con el apoyo de personal que se ocupe de sus necesidades. Esto ha conducido a realizar esfuerzos por aplicar la robótica y la automatización para cubrir estas tareas de asistencia.

Para que un robot de asistencia personal realice acciones que satisfagan determinadas necesidades fisiológicas, cuando no se cuenta con apoyo humano, debe de contar con las siguientes funciones: estimar su propia ubicación y la de los objetos, llegar al destino deseado evitando obstáculos, manipular los objetos, identificar las necesidades fisiológicas de los pacientes, y razonar sobre los objetos y operaciones necesarias para satisfacer estas necesidades fisiológicas.

Se estima que en las próximas décadas, en algunos países se duplicará el número de personas mayores de 80 años, y que se tendrá un déficit muy importante de profesionales de la salud. Si se pudiera desarrollar robots de cuidado personal capaces de realizar tareas de cuidado y asistencia básicas, la necesidad de apoyo de personal humano se reduciría significativamente. Sin embargo, a pesar de que se han desarrollado varios tipos de robots para brindar asistencia y cuidados, aún quedan retos importantes por superar, hasta lograr que los robots sean completamente autónomos y realicen de forma efectiva y segura estas tareas (Yang et al., 2019), (Miseikis et al., 2020).

El objetivo de este proyecto, es aportar en el desarrollo de este tipo de tecnologías, con la intención de que en un futuro estén al alcance de todos los sectores de la población que la requieran. En la organización Olin Robotics se está desarrollando un prototipo de exoesqueleto sobre una plataforma móvil, cuyo objetivo es asistir en el proceso de incorporación (sentado a parado)

* Autor para correspondencia: Jorge Enrique Luna Taylor (jorge.lt@lapaz.tecnm.mx)

Correo electrónico: L17310793@lapaz.tecnm.mx (Eloy Antonio Clemente Rosas), jorge.lt@lapaz.tecnm.mx (Jorge Enrique Luna Taylor), jo-se.gt@lapaz.tecnm.mx (José Luis Gómez Torres), isaac.vm@lapaz.tecnm.mx, (Isaac Villa Medina)

Historial del manuscrito: recibido el 16/05/2022, última versión-revisada recibida el 20/07/2022, aceptado el 09/08/2022, publicado el 05/10/2022. **DOI:** <https://doi.org/10.29057/icbi.v10iEspecial4.9262>



del usuario; además se considera que cuando el usuario no tenga el exoesqueleto colocado, este pueda tener funcionamiento autónomo para ir al centro de carga y cubrir algunas tareas de asistencia, como acercar vasos, platos o el móvil del usuario.

2. Antecedentes

En los últimos años, se han propuesto diferentes sistemas para tareas de asistencia y cuidado personal. Por ejemplo, en (Miseikis et al., 2020), presentan el desarrollo de una plataforma de robot móvil con un brazo multifuncional, combinando dispositivos de visión, audio, laser, ultrasónico y sensores mecánicos, controlados a través del sistema ROS (Robot Operating System) y algoritmos de aprendizaje profundo. Durante la pandemia de COVID-19, ajustaron el robot, en corto tiempo, para realizar tareas de desinfección y toma de temperatura.

En (Jishnu et al., 2020), proponen el diseño de un robot de asistencia, controlado a través de comandos de voz, para transportar objetos a cortas y largas distancias. La comunicación humano-robot se establece a través de dispositivos móviles vía Bluetooth. El robot cuenta con una cámara para la detección de los objetos, así como para calcular la distancia aproximada de estos, aplicando el algoritmo YOLO (You Only Look Once), basado en redes neuronales convolucionales.

Así mismo, en (Diddeniya et al., 2018), presentan el diseño de un robot que opera en ambientes de oficina, utilizando sensores para visión 3D, controlados a través del sistema ROS. Los comandos se transmiten por medio de un dispositivo móvil, que se conecta con una estación de trabajo central y el robot, vía Wi-Fi.

Independientemente de la aplicación, se han desarrollado una variedad de sistemas para localización de objetos y navegación autónoma. Por ejemplo, en (Ferreira, 2013), muestran el diseño de un vehículo para tareas de navegación autónoma, a partir de un mapa que crea de su entorno el propio sistema. El vehículo recopila la información a través de un sistema de visión, para identificar la trayectoria a seguir y detectar obstáculos. Para completar sus tareas, el vehículo es controlador de forma remota.

En (Purwanto et al., 2017), presentan el desarrollo de un sistema de navegación para interiores, a través de un sistema de visión multipunto a nivel del piso, para detectar obstáculos y áreas libres. Con esta información, aplican un sistema de inferencia difusa, para establecer la navegación del vehículo de forma autónoma.

Además, se han desarrollado propuestas enfocadas en el reconocimiento y estimación de la distancia de objetos, basadas en visión artificial. Por ejemplo, en (Emani et al., 2019), proponen un sistema de visión estéreo, que parte del reconocimiento de los objetos a través de una arquitectura MobileNet, con la cual obtienen recuadros que encierran a los objetos de interés. Posteriormente, estiman la distancia de los objetos detectados, aplicando una función de triangulación que toma como datos el desplazamiento en el eje X de los recuadros generados en las imágenes tomadas por ambas cámaras, la distancia entre las cámaras y la longitud focal de estas. En sus experimentos, utilizaron dos tipos de cámaras, la Web-cam rig y la cámara ZED, reportando un Error Cuadrático Medio (MSE) de 48.8 y 58.1, respectivamente para ambos tipos de cámaras.

En (Rahul and Nair, 2018), presentan un sistema que combina aprendizaje profundo y visión estéreo, para la detección y estimación de la distancia de objetos. Utilizan una red neuronal convolucional, para detectar e identificar los objetos, y después estiman su distancia construyendo una nube de puntos 3D utilizando el método de triangulación. El sistema se probó con una cámara estéreo ZED, logrando una precisión de reconocimiento del 84 % y un error promedio en la estimación de las distancias del 4.7 %.

En (Mukherjee et al., 2020), presentan un sistema basado en la plataforma ROS, para la detección y estimación de distancia de peatones. Para la detección, utilizan una cámara de visión estéreo ZED y la arquitectura de red convolucional YOLOv2. Generan múltiples recuadros delimitadores alrededor de los peatones, con las probabilidades de clase y los centroides de estos recuadros. Para la estimación de la distancia, toman las coordenadas de los centroides y extraen las coordenadas 3D de la nube de puntos correspondiente. Posteriormente, aplican por separado un sistema Leddar M16 y la información de profundidad de un sistema estéreo ZED, para calcular las distancias. Finalmente, aplican una combinación de ambos sistemas para mejorar la precisión de la estimación. Sus experimentos reportan un RMSE de 10.08 cm con el sistema Leddar, un RMSE de 15.79 cm con el sistema ZED y un RMSE de 6.94 cm con la fusión de ambos.

3. Desarrollo

En este proyecto se propone un sistema de reconocimiento y clasificación de objetos, así como la estimación de la posición relativa de estos, con respecto a una cámara montada sobre un robot móvil. En la Figura 1 se muestra el diagrama general del sistema propuesto. El sistema tiene dos etapas de operación: la primera etapa abarca el preprocesamiento de las imágenes para entrenar el modelo y el entrenamiento del mismo. Estas dos tareas se realizan *offline* en una computadora de escritorio. La segunda etapa corresponde al reconocimiento y clasificación de los objetos, así como la estimación de la posición relativa de estos. Estas tareas se realizan *online* en un sistema Raspberry Pi 4.

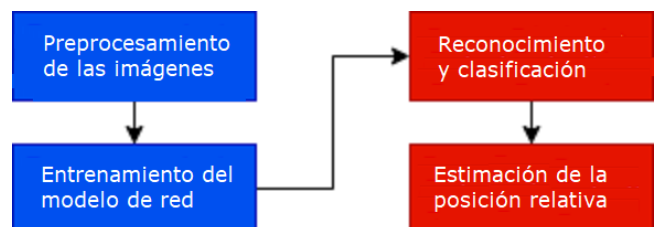


Figura 1: Diagrama general del funcionamiento del sistema.

3.1. Preprocesamiento de las imágenes

La preparación de las imágenes para el entrenamiento del modelo consistió en las siguientes tareas:

- Se definió un conjunto de 10 clases de objetos, a los cuales se les asignó un índice de clase y un color (ver Tabla 1). Estos objetos se encuentran dentro de un espacio real,

utilizado como centro de trabajo. Se consideran como objetos de interés aquellos a los cuales se pretende aproximar el robot en trabajos futuros. Las clases incluyen: conectores eléctricos, con la idea de que el robot se conecte de forma autónoma; celulares, platos y vasos para transportar; obstáculos comunes en ese entorno que el robot puede encontrar en su camino como herramienta, mesa, garrafón de agua, personas; otros obstáculos varios, poco comunes; y el resto de objetos como pared, piso, cortinas, etc., considerados como fondo.

Tabla 1: Definición de los objetos de interés.

Objeto	Color RGB	Índice
Fondo	(0, 0, 0)	0
Conector	(0, 128,128)	1
Celular	(128, 0, 0)	2
Plato	(0, 0, 128)	3
Vaso	(128, 128, 0)	4
Mesa	(0, 128, 0)	5
Personas	(192, 0, 0)	6
Herramienta	(128, 0, 128)	7
Garrafón de agua	(0,192,0)	8
Obstáculos	(0,0,192)	9

- Se capturaron un total de 397 imágenes, que incluyen a los objetos de interés, a no más de 2.5 metros de distancia. Las imágenes se guardaron originalmente en formato JPG. En la Tabla 2 se muestran las características básicas de la cámara utilizada. Para la captura de estas imágenes se configuró la cámara en 21fps, lo cual soporta un tamaño de las imágenes de hasta 8mp, sin embargo, al almacenar las imágenes se redujeron a un tamaño de 640x480 píxeles, el equivalente a un total de 307,200 píxeles.

Tabla 2: Especificaciones de la cámara.

Tipo de obturador	Rodante
Velocidad de fotogramas	21fps 8mp 30fps 1080p 120fps 720p
Ángulo de visión	75 grados (horizontal)

- Se delimitó manualmente el contorno de los objetos de interés presentes en cada imagen, utilizando el software *VIA VGG Image Anotator, Versión 2.0.11*¹ (ver Figura 2).
- Se generó un archivo JSON con la definición de los polígonos que representan el contorno correspondiente de cada objeto.
- Mediante una rutina escrita en Python, con la información del archivo JSON, se generaron imágenes en formato PNG, con los objetos segmentados con su color corres-

pondiente (ver Algoritmo 1). En la Figura 3 se muestra un ejemplo de una imagen segmentada y almacenada en formato PNG.

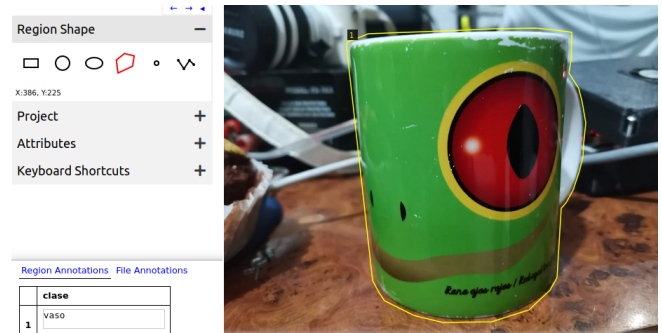


Figura 2: Delimitación del contorno de los objetos de interés.

Algoritmo 1: Generación de archivos PNG con imágenes segmentadas

```

entrada: Archivo JSON con los polígonos generados
salida : Archivos PNG con las imágenes segmentadas
for img in lista de imágenes en el archivo JSON do
  alto, ancho ← alto y ancho de img
  mascara ← genera nueva matriz tridimensional de
    (alto x ancho x 3)
  for poli in polígonos asociados a img en JSON do
    clase ← clase de objeto relacionado a poli
    (R, G, B) ← color predefinido para clase
    contorno ← crea matriz unidimensional vacía
    for (i, j) in lista de puntos de poli do
      | agrega punto (i, j) a contorno
    end
    for i in rango(ancho) do
      | for j in rango(alto) do
        | // función pointPolygonTest
        | if punto (i, j) se encuentra dentro del área
        | de contorno then
        |   mascara[j : i] ← (R, G, B) ;
        | end
      | end
    end
    almacena mascara como archivo PNG
  end
end
return

```

- Se escalaron las imágenes originales y las imágenes segmentadas a un tamaño de 320x240 píxeles.
- Se normalizaron las componentes de color de las imágenes originales (JPG) y se almacenaron con formato Numpy. Además, se convirtieron las componentes de color RGB de cada pixel de las imágenes segmentadas (PNG), a un valor único que corresponde a la clase del objeto al que pertenece el pixel, y se almacenaron las imágenes, también, con formato Numpy (ver Algoritmo 2).

¹<https://www.robots.ox.ac.uk/vgg/software/via/>



Figura 3: Segmentación de los objetos de interés.

Algoritmo 2: Generación de archivos Numpy

```

entrada: PATH de las carpetas de imágenes
salida : Archivos de imágenes con formato Numpy
PATH_ORI ← PATH de las imágenes JPG
PATH_SEG ← PATH de las imágenes PNG
for img_ori in PATH_ORI do
  | img_ori ← img_ori/255,0
  | almacena img_ori con formato Numpy
end
MAPA_COLOR ← [[0,0,0],[0,128,128],[128,0,0],[0,0,128],
[128,128,0],[0,128,0],[192,0,0],[128,0,128],[0,192,0],
[0,0,192]]
etiqueta ← crea tensor de 2563 en ceros
for i,map in enumeración(MAPA_COLOR) do
  | etiqueta[map[0]*256 + map[1])*256 + map[2]] ← i
end
for img_seg in PATH_SEG do
  | indice ← ((img_seg[:, :, 0] * 256 + img_seg[:, :,
  | 1]) * 256 + img_seg[:, :, 2])
  | img_seg ← etiqueta[indice]
  | // indice es una matriz bidimensional,
  | // esta operación devuelve otra matriz
  | // con la clase de objeto por píxel
  | almacena img_seg con formato Numpy
end
return

```

3.2. Entrenamiento del modelo

Con el objetivo de reconocer y clasificar los objetos en etapas posteriores, se implementó una estrategia de *Segmentación Semántica*, la cual consiste en clasificar las imágenes a nivel de píxel. Para esto, se utilizó una red neuronal con arquitectura tipo U-Net.

La arquitectura U-Net consta de dos bloques principales, el primer bloque, llamado *downsampling* o camino de contracción, funciona como extractor de características de las imágenes de entrada y disminuye la resolución de salida. Este bloque aporta a la red la capacidad de clasificación de los objetos. El segundo bloque, nombrado como *upsampling* o camino de expansión, es aproximadamente simétrico al *downsampling*, resultando la arquitectura con forma de U. Este bloque tiene como función propagar la información de las características extraídas a capas de más alta resolución, y aporta la capacidad de localización de los objetos clasificados. Cuando la resolución de sa-

lida corresponde a las dimensiones de las imágenes de entrada, es posible clasificar los objetos a nivel de píxel (Ronneberger et al., 2015).

En este proyecto, se utiliza el modelo ResNet18 (He et al., 2016) como bloque *downsampling*, al cual se le eliminaron las dos últimas capas, dado que estas se utilizan cuando el objetivo es clasificar a nivel de la imagen completa. Para realizar la función del bloque *upsampling*, al igual como se propone en (Ronneberger et al., 2015), se agregaron cuatro capas convolucionales transpuestas, combinadas con capas convolucionales comunes (ver Tabla 3).

Además de esta configuración, se evaluaron diferentes combinaciones, buscando un equilibrio entre precisión y el tiempo de inferencia sobre la Raspberry. Un incremento del número de capas o de la cantidad de canales por capa, mejora la calidad del reconocimiento, pero incrementa el tiempo de inferencia y viceversa. Con la configuración elegida, se logra un 90 % de precisión en el reconocimiento, con un tiempo de inferencia aproximado de seis segundos. Por ejemplo, reducir a la mitad el número de capas de ambos bloques (*down* y *upsampling*), el tiempo de inferencia se reduce a un segundo, pero con resultados de reconocimiento por debajo del 80 %.

Tabla 3: Arquitectura de la red neuronal utilizada.

Capas	Canales	Dimensión	K	Stride	Padding
Imagen	3	320x240			
Conv	64	160x120	7x7	2x2	3x3
Conv	64	160x120	3x3	1x1	1x1
Conv	64	160x120	3x3	1x1	1x1
Conv	64	160x120	3x3	1x1	1x1
Conv	128	80x60	3x3	2x2	1x1
Conv	128	80x60	3x3	1x1	1x1
Conv	128	80x60	3x3	1x1	1x1
Conv	128	80x60	3x3	1x1	1x1
Conv	256	40x30	3x3	2x2	1x1
Conv	256	40x30	3x3	1x1	1x1
Conv	256	40x30	3x3	1x1	1x1
Conv	256	40x30	3x3	1x1	1x1
Conv	512	20x15	3x3	2x2	1x1
Conv	512	20x15	3x3	1x1	1x1
Conv	512	20x15	3x3	1x1	1x1
Conv	512	20x15	3x3	1x1	1x1
Conv Trans.	256	40x30	2x2	1x1	2x2
Conv	256	40x30	3x3	1x1	1x1
Conv Trans.	128	80x60	2x2	1x1	2x2
Conv	128	80x60	3x3	1x1	1x1
Conv Trans.	64	160x120	2x2	1x1	2x2
Conv	64	160x120	3x3	1x1	1x1
Conv Trans.	64	320x240	2x2	1x1	2x2
Conv	10	320x240	3x3	1x1	1x1

En la Tabla 4 se muestran los parámetros del entrenamiento. Al igual que en el proceso para obtener la configuración de la arquitectura de la red, se evaluaron diferentes combinaciones de los parámetros, buscando disminuir el valor de la función de pérdida sobre los datos de entrenamiento y, principalmente, so-

bre los datos de prueba, sin incrementar de forma significativa el número de épocas requerido para obtener estos valores. Por ejemplo, aumentar la tasa de aprendizaje o el factor de decaimiento, mejora la velocidad del entrenamiento (menor número de épocas), pero incrementa la función de pérdida. Incrementar el número de épocas, con una configuración fija de los demás parámetros, reduce la función de pérdida sobre los datos de entrenamiento, pero sobre los datos de prueba, la función llega a un punto de inflexión, donde a partir de ese punto, se incrementa con cada época, esto es, el modelo se sobreentrena.

Tabla 4: Parámetros del entrenamiento.

Tamaño de lote	10
Número de épocas	100
Tasa de aprendizaje	0.0001
Factor de decaimiento	0.001
Función de pérdida	Entropía Cruzada
Función de optimización	Adam

El entrenamiento de la red se implementó en Python versión 3.9.7, utilizando las librerías Pytorch versión 1.9.0 y Torchvision versión 0.10.0. Se utilizaron 350 imágenes para entrenamiento y 47 para pruebas. En la Tabla 5 se muestran las características del equipo de cómputo donde se ejecutó el entrenamiento. La carga de los conjuntos de datos y el algoritmo de entrenamiento, es similar a como se explica en (Sensio, 2020).

Tabla 5: Características del equipo de cómputo para el entrenamiento.

Procesador	Intel Core i7-4790 3.60GHz x 8
Tarjeta gráfica	NVIDIA GeForce GTX TITAN X
Memoria RAM	16 GB
Disco duro	SSD 240 GB
Sistema operativo	Ubuntu 20.04 LTS

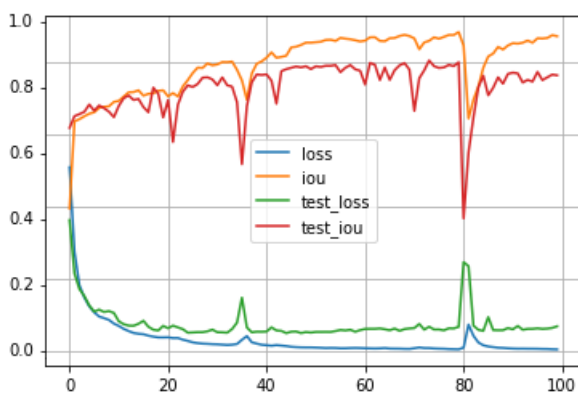


Figura 4: Estadística del entrenamiento.

En la Figura 4 se muestra la estadística del entrenamiento. El valor final de la función de pérdida sobre los datos de entrenamiento es de 0.00641 (color azul). Esta métrica está asociada con la cantidad de píxeles mal clasificados por la red. Para esto, el sistema compara la segmentación inferida por la red y la segmentación real que se introduce como dato. Un valor cero

indica que no hay píxeles mal clasificados. El factor IoU sobre los datos de entrenamiento es de 0.87202 (color naranja). Esta métrica corresponde a la relación entre el área de píxeles bien clasificados (coincidencias entre la segmentación real y la inferida), y el área total que abarcan ambas segmentaciones. Un valor uno indica una coincidencia exacta entre ambas segmentaciones.

Sobre los datos de prueba, la función de pérdida concluyó con un valor de 0.07024 (color verde) y el factor IoU de 0.76395 (color rojo). Los datos de prueba se utilizan durante el entrenamiento, únicamente para ver el comportamiento del modelo. Es decir, el conocimiento del error sobre estos datos, no se aplica en la mejora del modelo, como sí sucede con el error sobre los datos de entrenamiento. Por lo tanto, esta información es un parámetro más real de como se comportará el modelo con nuevos datos.

3.3. Reconocimiento y clasificación

El proceso de reconocimiento y clasificación de los objetos se implementó en lenguaje Python, sobre un sistema Raspberry Pi 4, equipado con un módulo de cámara. Para esto, se almacenó previamente el modelo de la red entrenada en formato Pytorch (.pt), y se cargó sobre la Raspberry.

Algoritmo 3: Reconocimiento de objetos sobre la Raspberry

entrada: Modelo de la red neuronal entrenada

salida : Imágenes con segmentación inferida

carga del *modelo* de la red neuronal

activa el *modelo* en modo predicción

configura y activa la cámara

repeat

imagen ← frame del buffer de la cámara

imagen ← *imagen* en formato Torch

imagen ← *imagen* con transferencia de la tercera dimensión hacia la primera dimensión

// Se transfieren los canales de color // como primera dimensión de la imagen

imagen ← *imagen* con nueva dimensión insertada como primera dimensión

// Se agrega una dimensión dado que la // red procesa por lote de imágenes

imagen ← *imagen.float()/255,0*

salida ← predicción de la *imagen* con el *modelo*

salida ← primera imagen de la dimensión cero de *salida*

// Se toma la primera (única) imagen

// del lote devuelto por la red

salida ← argumento máximo de la primera dimensión de *salida*

// Se reduce de tensor tridimensional

// a bidimensional con la clase infe-

// rida en cada pixel

despliega o almacena *salida* con formato PNG

until *pulsa tecla* < ESC >;

return

El proceso inicia con la captura de una imagen a través de la cámara, y se pasa a la red para su evaluación. La red devuelve un tensor con la predicción de la clase a la que pertenece cada pixel (ver Algoritmo 3).

Las dimensiones del tensor devuelto por la red son (1 x 10 x 240 x 320). La dimensión con subíndice cero corresponde al número de imágenes del lote procesado (se procesa una a la vez), la dimensión uno corresponde a la cantidad de clases por clasificar, las últimas dos dimensiones, corresponden al alto y ancho de la imagen. El valor de cada celda indica la probabilidad estimada por la red, con valor entre 0 y 1, de que el pixel que se encuentra en la fila y columna a la que pertenece la celda, sea de la clase con valor igual al subíndice de la dimensión uno de la celda. A este tensor se le elimina la dimensión cero, quedando ahora el número de clases como nueva dimensión cero.

Por ejemplo, en la Figura 5 se muestra un tensor con la predicción para tres clases (imagen superior). Se presentan únicamente los valores para la última columna (columna 319 de la dimensión dos). En color amarillo se muestran las celdas con el valor de probabilidad de que el pixel en la fila y columna correspondiente sea de la clase cero. En color verde, las celdas con la probabilidad de que el pixel sea de la clase uno, y en color naranja las celdas con la probabilidad de que el pixel sea de la clase dos.

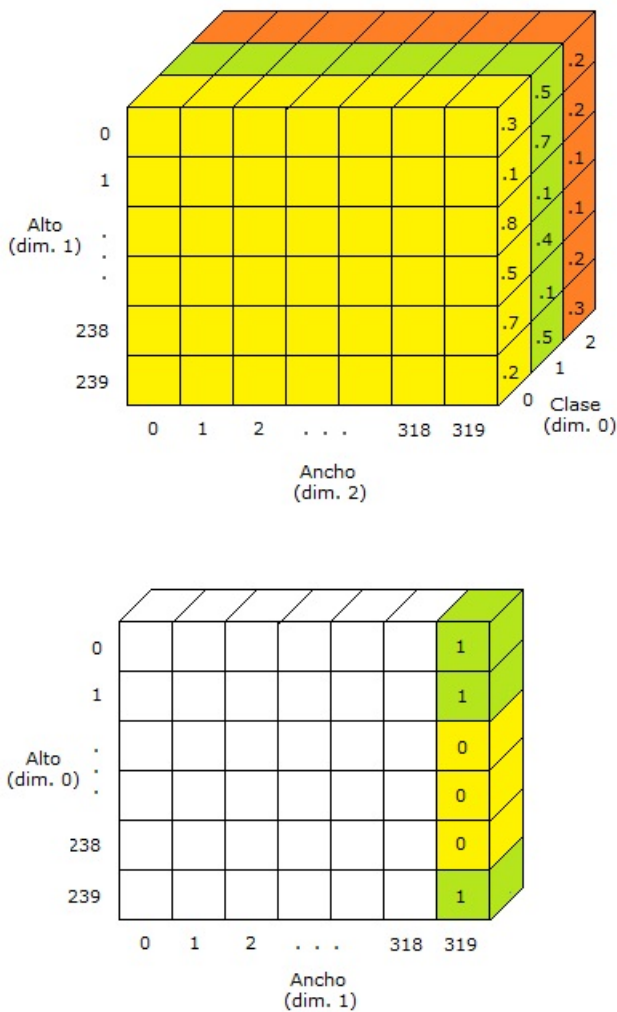


Figura 5: Ejemplo de tensor de probabilidades para tres clases (superior) y del tensor con predicción final de clases (inferior).

Para obtener la clase de cada pixel, se aplica la función

$torch.argmax(tensor_prob, axis = 0)$, sobre el tensor tridimensional, resultando un tensor bidimensional, donde el valor de cada celda corresponde a la clase inferida del pixel (1). En la imagen inferior de la Figura 5, se muestran las clases asignadas para la última columna, a partir del tensor de probabilidades de la imagen superior. Por ejemplo, el pixel de la fila cero y columna 319, tiene una probabilidad de 0.30 de pertenecer a la clase cero, 0.50 a la clase uno y 0.20 a la clase dos. Por lo que el valor asignado (clase con máxima probabilidad) es la clase uno. En la Figura 6 se muestra un ejemplo de una imagen de entrada a la red, y en la Figura 7, se muestra la imagen segmentada por el sistema.

$$clase_pixel(i, j) = k, \text{ tal que } tensor(k, i, j) = \max\{tensor(m, i, j) \mid 0 \leq m < len(clases)\} \quad (1)$$



Figura 6: Imagen original de entrada a la red.

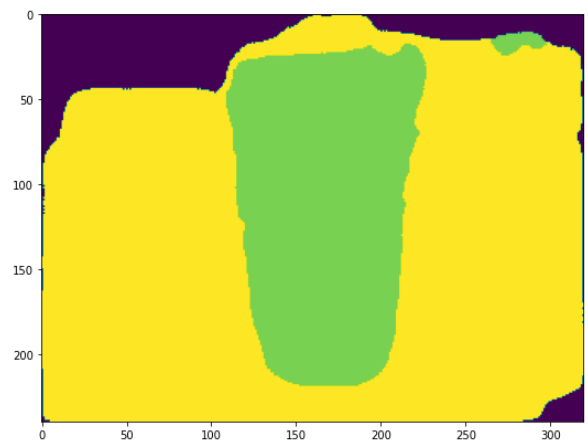


Figura 7: Segmentación inferida por la red.

3.4. Estimación de la distancia relativa

Una vez identificados y segmentados los objetos en la imagen, el siguiente paso es estimar su distancia relativa con respecto al robot. Para esto, se aplicó una técnica de visión estereoscópica, a partir de dos imágenes tomadas en posiciones diferentes de la cámara.

Esta estrategia, asume que a mayor distancia de los objetos con respecto a la cámara, menor será su desplazamiento al

comparar su posición en ambas imágenes. Y de forma correspondiente, a menor distancia de los objetos, mayor será su desplazamiento.

Para obtener la relación, entre el desplazamiento de los objetos en las imágenes y la distancia a la que se encuentran de la cámara, se capturaron ocho imágenes de tres puntos de control, posicionando la cámara a 50, 80, 100 y 150 cm de estos. Para cada una de estas distancias se capturaron dos imágenes, colocando la cámara a 6 cm de distancia entre una y otra toma. (ver Figura 8).

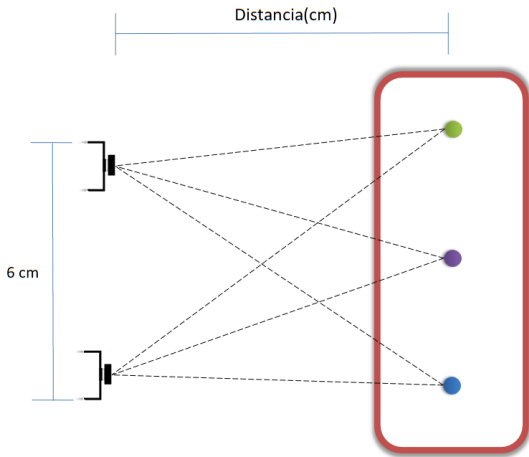


Figura 8: Captura de puntos de control desde dos posiciones distintas.

En la Tabla 6 se muestran la posiciones de los puntos de control en cada una de las imágenes capturadas. Las posiciones están dadas en píxeles sobre el eje X. Así mismo, se muestra el desplazamiento de los puntos entre las dos imágenes. Como desplazamiento asociado a cada una de estas distancias, se tomó el promedio de los tres puntos de control.

Tabla 6: Desplazamiento de los puntos de control.

		Posición en el eje X (píxeles)					
		Control uno		Control dos		Control tres	
		Pos.	Desp.	Pos.	Desp.	Pos.	Desp.
50 cm	Origen	320	73	197	74	76	72
	A 6 cm	247		123		4	
80 cm	Origen	320	47	239	48	160	48
	A 6 cm	273		191		112	
100 cm	Origen	320	38	254	39	190	39
	A 6 cm	282		215		151	
150 cm	Origen	320	26	276	27	232	26
	A 6 cm	294		249		206	

Una vez obtenidos los desplazamientos asociados a estas cuatro distancias, se aplicó análisis de regresión con el método *curve_fit* de la librería *scipy.optimize* de Python, para obtener la función que mejor describe esta relación. El método *curve_fit* aplica el algoritmo Levenberg-Marquardt (Levenberg, 1944), (Marquardt, 1963), para resolver el problema de ajuste de curvas por mínimos cuadrados (Scipy, 2022). Esto es, dado un conjunto de m pares (x_i, y_i) , aplica una combinación de los algoritmos Gauss-Newton y Descenso del Gradiente, para

encontrar los parámetros β del modelo de la curva $f(x, \beta)$, que minimicen la suma de los cuadrados de las diferencias (2).

$$\operatorname{argmin}_{\beta} \sum_{i=1}^m [y_i - f(x_i, \beta)]^2 \quad (2)$$

Para esto, se probaron diferentes funciones (ver Algoritmo 4), lineal (ver Figura 9), exponencial (ver Figura 10), e hiperbólica (ver Figura 11). Finalmente, la función hiperbólica dada en (3), resultó ajustarse mejor a los datos. Esta función hiperbólica, de la forma $y = k/x$, asume como variable independiente x , el desplazamiento de los objetos entre ambas imágenes, dado en píxeles; y calcula, como variable dependiente y , la distancia a la que se encuentran los objetos de la cámara, dada en centímetros.

$$\text{distancia(cm)} = 3836,15/\text{desplazamiento(px)} \quad (3)$$

Algoritmo 4: Análisis de Regresión

entrada: Distancias de los puntos de control y desplazamientos asociados

salida : Ecuaciones de las curvas que mejor se ajustan

define *func_lin*(x, m, b):

return $m \cdot x + b$

define *func_exp*(x, a, b):

return $a \cdot e^{b \cdot x}$

define *func_hip*(x, k):

return k/x

$x \leftarrow$ arreglo de distancias

$y \leftarrow$ arreglo de desplazamientos

linOpt \leftarrow *curve_fit*(*func_lin*, x, y)

$m, b \leftarrow$ *paramOpt*

genera y muestra gráfica lineal

expOpt \leftarrow *curve_fit*(*func_exp*, x, y)

$a, b \leftarrow$ *paramOpt*

genera y muestra gráfica exponencial

hipOpt \leftarrow *curve_fit*(*func_hip*, x, y)

$k \leftarrow$ *paramOpt*

genera y muestra gráfica hiperbólica

return

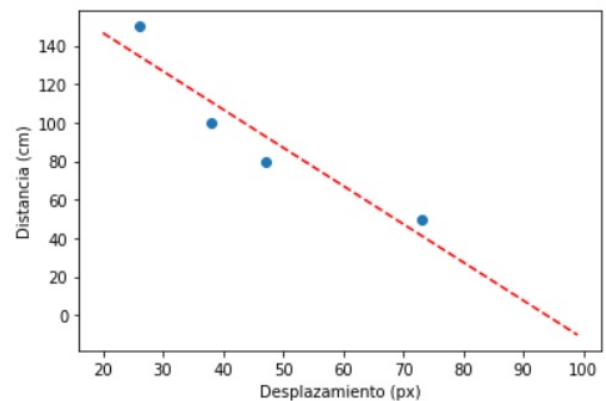


Figura 9: Regresión lineal.

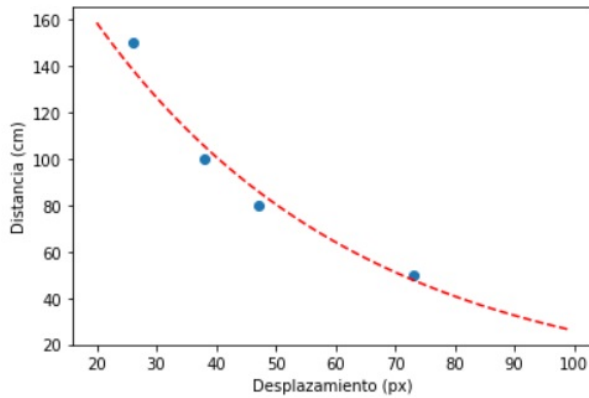


Figura 10: Regresión exponencial.

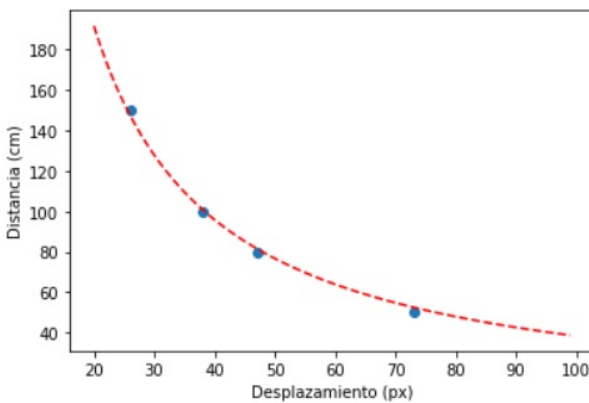


Figura 11: Regresión hiperbólica.

Con la función (3), se puede estimar la distancia relativa de los objetos con respecto al robot, a partir del desplazamiento de estos en las imágenes tomadas desde dos puntos diferentes. Para esto, una vez segmentadas las imágenes, se obtiene el contorno y el centroide de los objetos en ambas imágenes, utilizando la librería *OpenCV* de Python (ver Algoritmo 5). En la Figura 12 se muestra un ejemplo de la generación del contorno y centroide para un mismo objeto, en las dos imágenes capturadas.

El valor del desplazamiento se obtiene con el valor absoluto de la diferencia de la componente X de los centroides. En el ejemplo de la Figura 12, la componente X del centroide del objeto en la imagen superior es 227, y en la imagen inferior es de 176, resultando un desplazamiento de 51 píxeles. Aplicando la función (3), la distancia inferida es de 71 cm, lo cual es cercano a la distancia real, que es de 70 cm.

Algoritmo 5: Cálculo de Distancia Relativa

entrada: Modelo de la red neuronal entrenada, *clase* del objeto de interés
salida : Distancia relativa estimada del objeto de interés
 carga del *modelo* de la red neuronal
 activa el *modelo* en modo predicción
 configura y activa la cámara uno
 configura y activa la cámara dos
repeat
 *imagen*₁ ← frame del buffer de la cámara uno
 *imagen*₂ ← frame del buffer de la cámara dos
 *salida*₁ ← Proceso de reconocimiento con *imagen*₁
 // reconocimiento con Algoritmo 3
 *salida*₂ ← Proceso de reconocimiento con *imagen*₂
 *salida*₁ ← filtrado de píxeles de la *clase* de interés de *salida*₁
 *salida*₂ ← filtrado de píxeles de la *clase* de interés de *salida*₂
 *contorno*₁ ← identificación de contorno en *salida*₁
 // Función *findContours* de *OpenCV*
 *contorno*₂ ← identificación de contorno en *salida*₂
 *M*₁ ← cálculo de momentos del *contorno*₁
 // Función *moments* de *OpenCV*
 // $M_{i,j} = \sum_x \sum_y x^i y^j I(x,y)$
 *M*₂ ← cálculo de momentos del *contorno*₂
 *centr*_{1x} ← $M_{-1,0}/M_{0,0}$
 *centr*_{2x} ← $M_{-2,0}/M_{2,0}$
 desplazamiento ← $abs(centr_{2x} - centr_{1x})$
 distancia ← $3836,15/desplazamiento$
 despliega *distancia*
until pulsa tecla < ESC >;
return

Un problema detectado al aplicar esta estrategia es que, si en una o en ambas imágenes, el objeto de interés se encuentra en un extremo, parcialmente fuera del campo de visión de la cámara, no es probable que el centroide corresponda al mismo punto de referencia sobre el objeto en ambas imágenes. Es decir, no se estaría midiendo el desplazamiento del mismo punto (o puntos cercanos), por lo tanto, no sería un indicador confiable de cuanto se desplaza el objeto. Por ejemplo, en la Figura 12, tanto en la imagen superior como inferior, se obtiene el centroide aproximadamente en el centro de la taza completa. En cambio, el centroide de la taza de la Figura 13 se encuentra cargado al lado derecho de la taza (si esta se viera completa), por lo que el desplazamiento de este centroide, con respecto a cualquiera de los dos centroides de la Figura 12, no representaría una buena aproximación del desplazamiento de la taza.

La propuesta para estos casos, es utilizar las coordenadas del borde del objeto que se encuentra del lado contrario al extremo por donde sale parcialmente de la imagen. Esto es, si el objeto sale parcialmente por el lado izquierdo de la imagen, se toma la componente X máxima (contorno derecho) del objeto (círculo amarillo en la Figura 13). Si el objeto sale parcialmente por el extremo derecho, se toma la componente X mínima. Los resultados de aplicar esta estrategia se presentan en la siguiente sección.

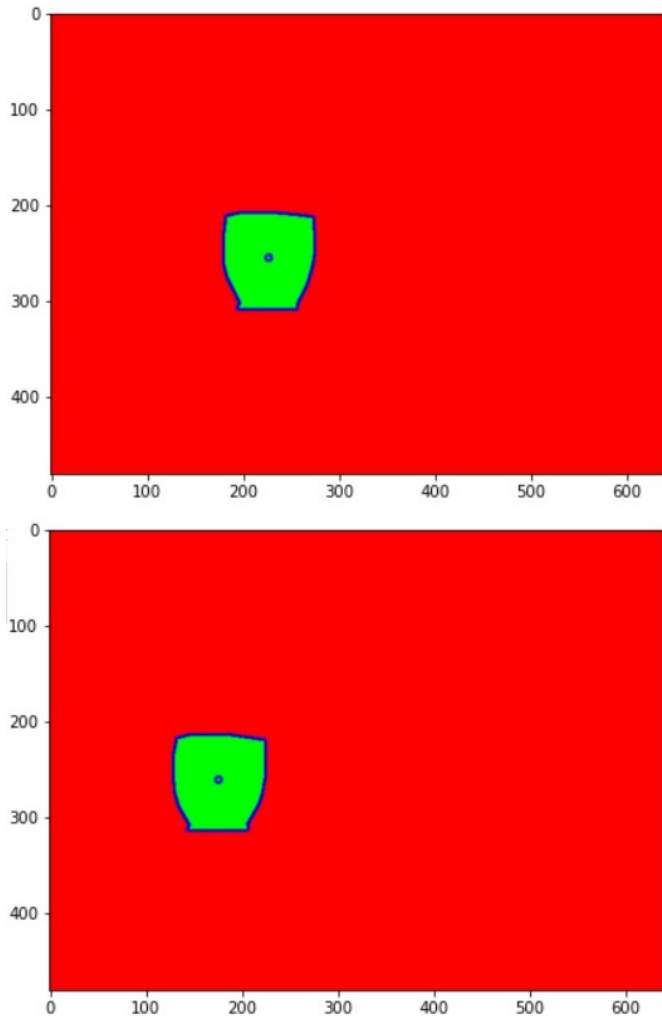


Figura 12: Ejemplo de la generación de contornos y obtención de centroides.

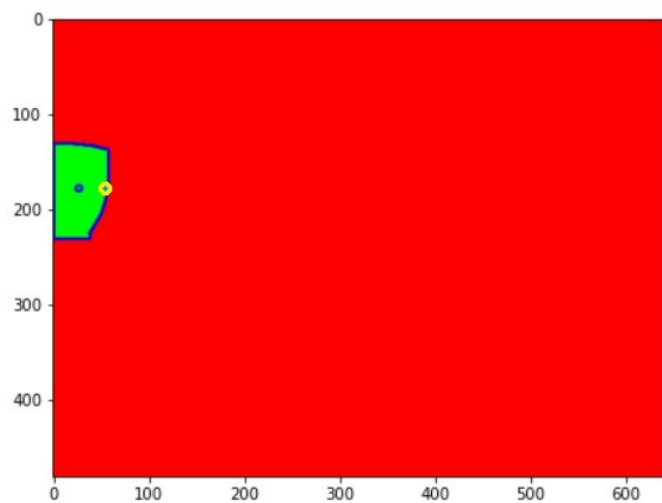


Figura 13: Objeto parcialmente fuera del campo de visión.

4. Experimentos y resultados

4.1. Pruebas de reconocimiento y clasificación

Las pruebas consistieron en aplicar el sistema de reconocimiento y clasificación sobre 47 imágenes que contienen un total de 127 ocurrencias de los objetos de interés. En la Figura 14 se muestran ejemplos de las pruebas realizadas. En la primera columna se muestran las imágenes originales, en la segunda columna se muestran las imágenes con la segmentación definida de forma manual (considerada como real), y en la tercera columna se muestran las imágenes con la segmentación inferida por la red.

Se puede observar que la segmentación inferida es similar a la segmentación real. Por ejemplo, en la Figura 14-A, en ambas imágenes segmentadas se muestran en color amarillo los píxeles que se etiquetaron como parte de la mesa, o como equipo electrónico sobre esta. Se etiquetaron los píxeles del celular en color azul turquesa, y en color violeta los píxeles del fondo, que incluye piso, pared, cortinas y otros objetos no considerados de interés. Los colores asignados durante las pruebas a las diferentes clases de objetos no corresponden necesariamente con el color asignado en la preparación de las muestras, sin embargo, el valor de la etiqueta asignada sí corresponde.

En algunos casos se detectaron falsos positivos, por ejemplo, en la Figura 14-A se etiquetó el brillo sobre el borde del celular como parte de un plato. Un caso similar ocurrió en el reconocimiento del tomacorriente de la Figura 14-E. Se detectaron, también, casos de falsos negativos, por ejemplo, en la Figura 14-C no se reconoció una de las patas de la mesa. Para mejorar el reconocimiento, y disminuir estos casos de errores, se propone reentrenar la red con una mayor cantidad de imágenes de muestra, considerando mayor variación en la orientación de las tomas, diferentes condiciones de brillo, diferentes distancias y generar la segmentación manual de forma más exacta.

En la Tabla 7 se muestra la estadística de los resultados de las pruebas. El número de objetos reconocidos y clasificados con la clase correcta es de 116 (positivos correctos), la cantidad de objetos identificados con una clase no correcta es de 12 (falsos positivos) y el número de objetos presentes en las imágenes, no identificados por el sistema, es de 11 (falsos negativos). En esta estadística no se incluye el fondo como objeto de interés.

Tabla 7: Pruebas de reconocimiento.

Positivos correctos	116
Falsos positivos	12
Falsos negativos	11
Precisión	90.6 %
Sensibilidad	91.3 %



Figura 14: Ejemplos de pruebas de reconocimiento y clasificación.

Para un análisis de la predicción al nivel de clases, en la Figura 15 se muestra el mapa de calor de los resultados de la clasificación. En este caso si se incluye el fondo como un objeto más, para mostrar los casos en que un objeto de interés se reconoció como fondo y viceversa. Dado que en las imágenes de prueba resultó un bajo número de garrafones y de obstáculos, estos no se incluyen en el mapa de calor.

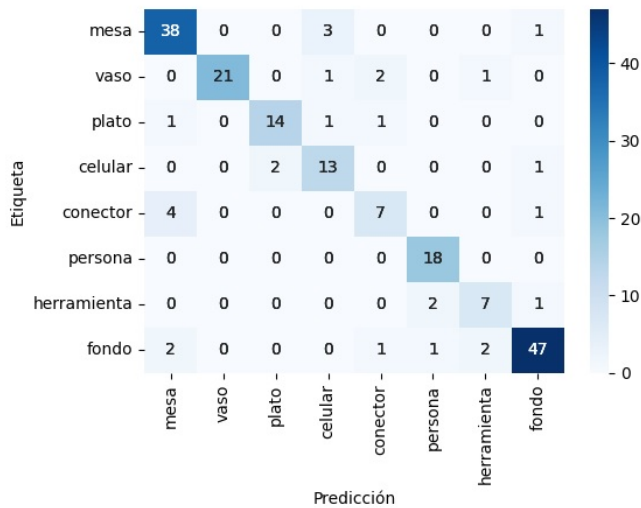


Figura 15: Mapa de calor de las pruebas de clasificación.

Se observa que uno de los errores más persistentes de la red, es en el reconocimiento de los conectores, los cuales confunde como parte de una mesa. Sin embargo, similar al caso de la Figura 14-E, en los cuatro casos reportados en el mapa, la confusión es parcial, es decir, reconoce el conector, pero parte de este lo confunde como parte de una mesa. Lo mismo sucede con el celular de Figura 14-A, identifica correctamente el celular, pero el borde con brillo de este lo confunde como parte de un plato.

4.2. Pruebas de estimación de distancias relativas

Para estas pruebas se colocaron objetos en diferentes posiciones con distancias conocidas y se aplicó la estrategia propuesta. Por cada posición de los objetos se capturaron dos imágenes con las cámaras separadas entre sí a una distancia de 6 cm. Una vez obtenidas las imágenes, se generó el contorno y centroide de los objetos de prueba, se calculó el desplazamiento de los centroides, se estimó la distancia de los objetos respecto a las cámaras y se obtuvo el error de la estimación.

Por ejemplo, en la Figura 16 se muestran las imágenes que corresponden a la prueba número uno reportada en la Tabla 8. Ambas imágenes (superior e inferior) se tomaron con la cámara a una distancia de 150 cm de la taza. Una vez segmentadas las imágenes, el centroide de la taza de la imagen superior resultó en la coordenada $x = 155$ y el centroide de la taza de la imagen inferior en $x = 130$, que corresponde a un desplazamiento del centroide de 25 cm. Aplicando la función (3) se obtiene una distancia estimada de 153 cm y un error respecto a la distancia real de 3 cm.

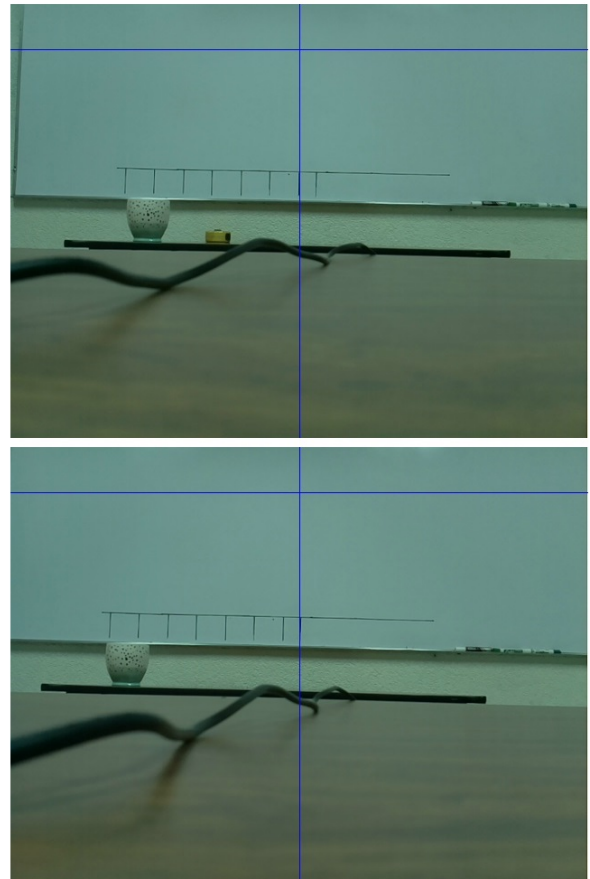


Figura 16: Imágenes de la prueba número uno de estimación de distancias.

En la quinta columna de la Tabla 8, se muestra el error de estimación al considerar únicamente los centroides para el cálculo del desplazamiento. Se puede observar que el error para las pruebas 8 y 10 es bastante elevado. Esto es debido a que en estos casos, el objeto se sale parcialmente de la imagen, por lo que los centroides calculados no corresponden al mismo punto de referencia del objeto en ambas imágenes. En la columna ocho se muestra el error para estas dos pruebas, con el cálculo del desplazamiento ajustado con base en el borde izquierdo del objeto. De esta forma, el error final promedio en la estimación de las distancias es de 4.6 cm, lo cual representa un 95,2 % de precisión, con una raíz del error cuadrático medio (RMSE) de 6.1 cm (ver Tabla 9).

Tabla 8: Pruebas de estimación de las distancias de los objetos.

Prueba	Dist. Real (cm)	Desplaz. (centr.)	Dist. Estim. (cm)	Error (cm)	Desplaz. (borde)	Dist. Estim. (cm)	Error (cm)
1	150	25	153	3			
2	150	23	166	16			
3	120	31	123	3			
4	120	31	123	3			
5	100	36	106	6			
6	100	38	101	1			
7	70	51	75	5			
8	70	31	124	54	53	68	2
9	50	72	53	3			
10	30	69	56	25	111	26	4

Tabla 9: Resultados de las pruebas de estimación de distancias.

MSE	37,4
RMSE	6,1
Distancia real promedio	96 cm
Error promedio	4,6 cm
Desviación estándar	4,2 cm
Precisión de la estimación	95,2 %

En la Figura 17 se muestra gráficamente la distancia de los objetos inferida por el sistema (componente Y de los puntos naranjas), a partir del desplazamiento de los centroides. Se muestra también la distancia real de los objetos (componente Y de los puntos azules). La componente X de cualquier punto (azul o naranja), corresponde al desplazamiento.

En la Figura 18 se muestran las distancias, una vez que se ajustó el desplazamiento de los objetos que salen parcialmente de la imagen (pruebas 8 y 10). Para estos objetos se utilizó el desplazamiento de sus bordes izquierdos, en lugar de sus centroides. Se puede observar que las distancias estimadas se ajustan mejor a las distancias reales.

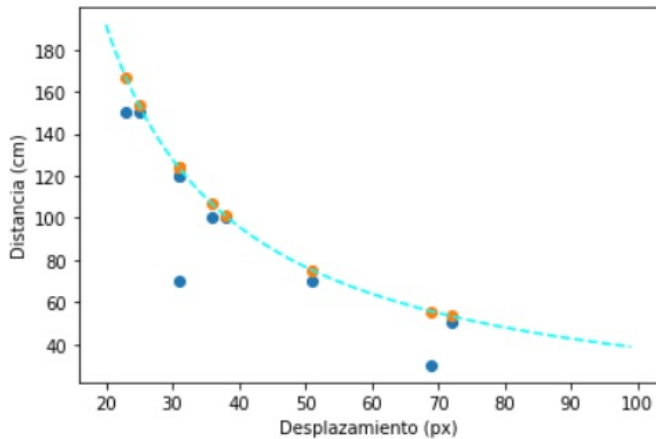


Figura 17: Estimación de las distancias con base en centroides.

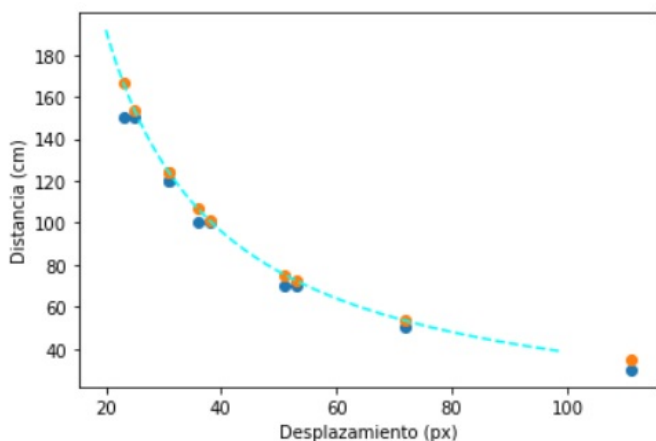


Figura 18: Estimación de las distancias con base en centroides y bordes.

5. Conclusiones

En este trabajo se propone un sistema para el reconocimiento y clasificación de objetos basado en redes neuronales convolucionales, así como la estimación de la posición relativa de

estos con respecto a una cámara montada sobre un robot móvil, aplicando una técnica de visión estereoscópica.

En las pruebas de reconocimiento se obtuvo un 90,6 % de precisión, mientras que en las pruebas de estimación de distancia relativa, se logró una precisión del 95,2 %. Consideramos que es posible mejorar de forma significativa la precisión del reconocimiento de los objetos, utilizando un mayor y más variado conjunto de muestras de entrenamientos y a través de una segmentación manual más precisa.

El objetivo final de este proyecto, para próximas etapas de desarrollo, es integrar los sistemas de reconocimiento y estimación de distancias propuestos, sobre un robot de cuidados y asistencia básica.

Agradecimientos

Agradecemos a la Dirección de Posgrado, Investigación e Innovación del Tecnológico Nacional de México, por el financiamiento de este trabajo, a través del proyecto 15319.22-P.

Referencias

- Diddeniya, S. I. A. P., Adikari, A. M. S. B., Gunasinghe, H. N., Silva, P. R. S. D., Ganegoda, N. C., and Wanniarachchi, W. K. I. L. (2018). Vision based of office assistant robot system for indoor office environment. *3rd International Conference on Information Technology Research (ICITR)*.
- Emani, S., Soman, K., Sajith, V., and Adarsh, S. (2019). Obstacle Detection and Distance Estimation for Autonomous Electric Vehicle Using Stereo Vision and DNN. In: Wang, J., Reddy, G., Prasad, V., Reddy, V. (eds) *Soft Computing and Signal Processing. Advances in Intelligent Systems and Computing*, 898:639–648.
- Ferreira, L. (2013). Localization and navigation in an autonomous vehicle. *Universidade de Aveiro Departamento de Electrónica, Telecomunicaciones e Informática*.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.
- Jishnu, K., Indu, V., Ananthkrishnan, K., Amith, K., Reddy, P., and Pramod, S. (2020). Voice controlled personal assistant robot for elderly people. *Proceedings of the Fifth International Conference on Communication and Electronics Systems (ICCES 2020)*, pages 269–274.
- Levenberg, K. (1944). A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, 2(2):164–168.
- Marquardt, D. (1963). An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, 11(2):431–441.
- Miseikis, J., Caroni, P., Duchamp, P., Gasser, A., R. Marko, N. M., Zwilling, F., Castelbajac, C., Eicher, L., Fruh, M., and Fruh, H. (2020). Lio-a personal robot assistant for human-robot interaction and care applications. in *IEEE Robotics and Automation Letters*, 5:5339–5346.
- Mukherjee, A., Adarsh, S., and Ramachandran, K. I. (2020). Ros-based pedestrian detection and distance estimation algorithm using stereo vision, leddar and cnn.
- Purwanto, D., Rivai, M., and Soebhakti, H. (2017). Vision-based multi-point sensing for corridor navigation of autonomous indoor vehicle. *International Conference on Electrical Engineering and Computer Science (ICECOS)*, pages 67–70.
- Rahul and Nair, B. B. (2018). Camera-Based Object Detection, Identification and Distance Estimation. *2018 2nd International Conference on Micro-Electronics and Telecommunication Engineering (ICMETE)*, pages 203–205.
- Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention*, 9351.
- Scipy (15 julio 2022). `scipy.optimize.curve_fit`. API reference. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html.
- Sensio, J. (3 octubre 2020). Visión Artificial - Segmentación Semántica. Sensio Inteligencia Artificial. https://juansensio.com/blog/050_cv_segmentacion.
- Yang, G., Wang, S., and Yang, J. (2019). Desire-driven reasoning for personal care robots. in *IEEE Access*, 7:75203–75212.