UNIVERSIDAD SANTO TOMÁS

SISTEMA DE IDENTIFICACIÓN DE OBJETOS EN ESPACIOS CERRADOS BASADO EN SEGMENTACIÓN SEMÁNTICA

Realizado por

Angela Maria Rojas Guayambuco

Angela Maria Sarria Arteaga

Proyecto de Grado presentado en cumplimiento del requisito
para optar por el grado de Ingeniería Electrónica

Grupo de Investigación GED (Grupo de Estudio y Desarrollo en Robótica)
Facultad de Ingeniería Electrónica
División de Ingenierías

Noviembre de 2023

SISTEMA DE IDENTIFICACIÓN DE OBJETOS EN ESPACIOS CERRADOS BASADO EN SEGMENTACIÓN SEMÁNTICA

Realizado por

Angela Maria Rojas Guayambuco
Angela Maria Sarria Arteaga

Proyecto de Grado presentado en cumplimiento del requisito
para optar por el grado de Ingeniería Electrónica

Dirigido por

PhD. Juan Manuel Calderon Chávez

Grupo de Investigación GED (Grupo de Estudio y Desarrollo en Robótica)
Facultad de Ingeniería Electrónica
División de Ingenierías

Noviembre de 2023

Autoridad de la Universidad

RECTOR GENERAL
Fray José Gabriel Mesa Angulo, O.P.

VICERRECTOR ADMINISTRATIVO Y FINANCIERO GENERAL
Fray Wilson Fernando Mendoza Rivera, O.P.

VICERRECTOR ACADÉMICO GENERAL
Fray Eduardo González Gil, O.P.

SECRETARIA GENERAL
Ingrid Lorena Campos Vargas

DECANO DIVISIÓN DE INGENIERÍAS
Fray Javier Antonio Hincapíe Ardila, O.P.

SECRETARIA DE DIVISIÓN
E.C. Luz Patricia Rocha Caicedo.

DECANO FACULTAD DE INGENIERÍA ELECTRÓNICA
Ing. Carlos Andrés Torres Pinzón

Nota de aceptación

_____
_____
_____
_____
_____
_____

_____

Firma del autor

_____

Firma del jurado

_____

Firma del jurado

BOGOTÁ D.C. ———– DE 2023

Advertencia

La Universidad Santo Tomás no se hace responsable de las opiniones y conceptos expresados en el trabajo de grado, solo velará por qué no se publique nada contrario al dogma ni a la moral católica y porque el trabajo no tenga ataques personales y únicamente se vea el anhelo de buscar la verdad científica.

Capítulo III –Art. 46 del Reglamento de la Universidad Santo Tomás.

Dedicatoria

Dedicamos esta tesis a todos aquellos que estuvieron presentes en el camino académico, tanto profesores como compañeros, quienes han sido fundamentales en el cumplimiento de este sueño.

A nuestros padres, les agradecemos infinitamente por su comprensión, apoyo incondicional. Por enseñarnos a enfrentar las adversidades sin perder nunca la dignidad ni desfallecer en el intento. Han sido nuestras guías y modelos para seguir, brindándome los valores, principios y la fortaleza necesaria para llegar hasta aquí.

Agradecimientos

Queremos agradecer a nuestro director Juan Manuel Calderon, a nuestro asesor Gerson David Cruz, a nuestra familia, amigos, profesores y todas las personas que influyeron en la realización de este proyecto de grado. Sus apoyos, consejos y contribuciones fueron fundamentales para el éxito del presente proyecto. Estamos sinceramente agradecidas por su confianza y apoyo en este camino hacia la culminación de este ciclo académico.

Índice general

Resumen

Este proyecto propone realizar el reconocimiento de objetos en entornos cerrados basado en segmentación semántica utilizando redes neuronales profundas. Para lograr esto, se han seleccionado dos arquitecturas de referencia ampliamente utilizadas en el campo de la visión: YOLO (*You Only Look Once*, YOLOv7) y Mask R-CNN.

La elección de la arquitectura YOLOv7 para la detección de objetos se debe a su capacidad para identificar objetos de manera eficiente en tiempo real. YOLO utiliza una única iteración para detectar objetos en una imagen, lo que la hace especialmente adecuada para aplicaciones donde la velocidad es un factor crítico.

Por otro lado, la arquitectura Mask R-CNN se seleccionó para abordar la tarea de la segmentación semántica. Esta permite asignar una máscara a cada objeto detectado, lo que brinda información detallada de la forma precisa de cada objeto en la imagen.

Ambas arquitecturas, YOLO y Mask R-CNN, se han entrenado y evaluado utilizando la reconocida base de datos COCO (*Common Objects in Context*). COCO ofrece una amplia variedad de imágenes etiquetadas y anotadas, lo que permite entrenar las redes neuronales en un conjunto diverso de categorías de objetos y contextos.

Abstract

This project proposes to perform object recognition in indoor environments based on semantic segmentation using deep neural networks. To achieve this, two reference architectures widely used in the vision field have been selected: YOLO (YOLOv7) and Mask R-CNN.

The choice of the YOLOv7 architecture for object detection is due to its ability to efficiently identify objects in real time. YOLO uses a single pass through the neural network to detect objects in an image, making it especially suitable for real-time applications where speed is a critical factor.

On the other hand, the Mask R-CNN architecture was selected to address the task of semantic segmentation. It is used for mask assigment to each detected object, which provides detailed information on the precise shape of each object in the image.

Both architectures, YOLO and Mask R-CNN, have been trained and evaluated using the well-known COCO (*Common Objects in Context*) database. COCO provides a wide variety of labeled and annotated images, allowing neural networks to be trained on a diverse set of object categories and contexts.

Índice de figuras

Índice de cuadros

Glosario

- Arquitecturas de red neuronal: Estructuras o diseños específicos de redes neuronales utilizados para tareas de aprendizaje automático.
- Datos de entrenamiento (Dataset): Conjunto de datos utilizado para entrenar un modelo de aprendizaje automático.
- Detección de objetos: Proceso de identificar y localizar objetos específicos dentro de una imagen o video.
- Eficiencia: La capacidad de la red para lograr un rendimiento óptimo y realizar sus tareas de manera efectiva, utilizando los recursos disponibles de manera eficiente.
- FPS (Freame per Second): Medida que indica la cantidad de cuadros o imágenes que un sistema puede procesar por segundo. Se utiliza para evaluar la velocidad de procesamiento en tareas de visión por computadora.
- Parámetros: Variables ajustables dentro de un modelo de red neuronal que influyen en su rendimiento y capacidad de aprendizaje.
- Píxel: Es la menor unidad homogénea en color que forma parte de una imagen digital.
- Precisión: Una medida que evalúa la proporción de píxeles o elementos clasificados correctamente por un modelo en relación con el total de píxeles o elementos evaluados.
- Preprocesamiento: El conjunto de operaciones y técnicas que se aplican a los datos de entrada antes de que se utilicen en un algoritmo o modelo de aprendizaje automático. En el contexto de la segmentación semántica, puede incluir la normalización, el escalado o la mejora de la calidad de las imágenes.
- Postprocesamiento: Proceso que se realiza después de obtener los resultados de un modelo, con el fin de mejorar la precisión o calidad de dichos resultados. Puede incluir técnicas como filtrado de ruido, suavizado o eliminación de falsos positivos.

- Recursos computacionales: Los recursos como potencia de procesamiento, capacidad de almacenamiento y memoria disponibles en un sistema informático para llevar a cabo tareas y operaciones.
- Red neuronal: Un sistema de procesamiento de información inspirado en el funcionamiento del cerebro, compuesto por un conjunto interconectado de nodos llamados neuronas artificiales que trabajan en conjunto para realizar tareas específicas de aprendizaje y procesamiento de datos.
- ResNet: Una arquitectura de red neuronal convolucional profunda que utiliza conexiones residuales para facilitar el entrenamiento de redes más profundas y mejorar su rendimiento.

Capítulo 1

Introducción

1.1. Planteamiento del Problema

La visión por computadora es una parte importante de la inteligencia artificial, ya que se ocupa de la teoría y algoritmos que permiten automatizar el proceso de percepción visual por medio de la implementación de tareas de diferente nivel [1]. Sin embargo, a pesar de los grandes progresos alcanzados, especialmente en el ámbito de reconocimiento facial, aún no están resueltos del todo problemas elementales como por ejemplo el reconocimiento general de objetos o el análisis interpretativo de escenas [2].

Cuando se refiere a la identificación y reconocimiento, cobra relevancia el proceso de segmentación. La segmentación semántica es una tarea esencial en el campo de la visión computacional, la cual consiste en dividir una imagen en varias regiones para realizar una clasificación por pixel y asignar una categoría a cada uno de estos. Es por eso que se considera de gran importancia estudiar la segmentación como paso clave en el procesamiento de imágenes.

En la actualidad una de las corrientes principales de la investigación son los métodos de segmentación semántica basados en el aprendizaje profundo, estos métodos pueden llegar a obtener buenos resultados en el proceso de segmentación semántica de imágenes parciales, sin embargo algunos de estos resultados no son evidentes en condiciones de interior, debido a que la estructura de la imagen en las escenas de interior es compleja, además de que se presentan problemas, tal como la iluminación desigual, la oclusión mutua en los objetos y similitudes en el color y la estructura de dichos objetos [3].

Se ha observado que ante esta problemática los investigadores han optado por utilizar la característica de profundidad para resolver este problema. En las escenas de interior, el color de la imagen RGB difumina los límites entre los objetos y ocasiona una pérdida grande de información espacial, no obstante, los datos de profundidad pueden proporcionar la relación geométrica correspondiente a la imagen RGB y permiten conservar dicha información espacial [4].

De acuerdo a la problemática expuesta anteriormente, se plantea la siguiente pregunta de investigación:

¿Cómo desarrollar un sistema de segmentación semántica para la identificación de objetos específicos al interior de un entorno conocido como el laboratorio de robótica de la Universidad Santo Tomás, sede principal haciendo uso de redes neuronales profundas?

1.2. Objetivos

1.2.1. Objetivo General

Desarrollar un sistema de identificación de objetos específicos en un entorno cerrado conocido basado en segmentación semántica usando redes neuronales convolucionales.

1.2.2. Objetivos Específicos

- Revisar información acerca de los diferentes tipos de redes neuronales convolucionales, algoritmos de segmentación semántica y métricas para la evaluación de este tipo de algoritmos, haciendo uso de los recursos y bases de datos especializados brindados por la Universidad para obtener una panorámica general de dichos algoritmos.
- Determinar el algoritmo de segmentación semántica que se adecue al sistema de visión para reconocer objetos específicos en un espacio cerrado conocido (Laboratorio de robótica Facultad de Ingeniería Electrónica - Universidad Santo Tomás, sede principal).
- Implementar el algoritmo de segmentación semántica definido previamente haciendo uso del lenguaje de programación Python y librerías especializadas para dicho fin.
- Evaluar el desempeño del sistema para validar el correcto funcionamiento mediante métricas de interés definidas previamente.

1.3. Justificación

En los últimos años, la inteligencia artificial, junto con los avances conseguidos en el campo de la visión por computador, han permitido ampliar las técnicas y algoritmos usados para realizar las tareas dentro del aprendizaje profundo. Sin embargo, en el campo de la visión, los desafíos consisten en llevar a cabo con mayor precisión tareas como clasificación, localización y asignación de etiquetas semánticas a los objetos que contiene una imagen [5].

Por otra parte, ha surgido un creciente interés en el campo de reconocimiento de escena, lo que ha impulsado la necesidad de comprender imágenes complejas. Para llevar a cabo este proceso se utiliza la segmentación semántica, una tarea de alto nivel que busca lograr una mejor compresión de la escena al asignar una etiqueta semántica a cada pixel de una imagen, permitiendo identificar los elementos que la componen. No obstante, es importante tener en cuenta que la segmentación no se lleva a cabo de manera aislada en el campo de la visión por computadora [5]. A pesar de que esta propuesta de trabajo de grado está orientada a la identificación de objetos específicos en interiores cerrados, cabe destacar que este tipo de herramienta también se usan en campos como la conducción autónoma, la localización y navegación de robots, reconocimiento facial, realidad virtual, entre otras.

A partir de lo mencionado, se identifican dos aspectos cruciales que influyen en la calidad en los resultados de una segmentación semántica. El primero implica diseñar una representación de características que permitan distinguir los objetos de diferentes clases. El segundo se refiere a la forma en que se reutiliza la información contextual para asegurar la consistencia entre las etiquetas de los píxeles coherentemente. En consecuencia, algunos de los modelos computacionales que han sido ampliamente utilizados en este tipo de tareas están basados en arquitecturas de aprendizaje profundo, específicamente de las redes neuronales convolucionales (CNN) [5].

Finalmente, se resalta que esta propuesta de trabajo de grado dentro de la Facultad de Ingeniería Electrónica en la Universidad Santo Tomás, sede principal, es uno de los primeros en trabajar con esta temática, lo que permitirá aplicar los conocimientos de inteligencia artificial, en especial la rama de visión por computadora para la búsqueda de soluciones de problemas reales e incentivar el uso de la segmentación semántica como una forma de interacción.

1.4. Impacto Social

Hoy en día la visión artificial cumple un rol importante en diferentes ámbitos del desarrollo tecnológico como lo son tareas de reconocimiento, detección de objetos, análisis de imágenes médicas, entre otros. La segmentación semántica, como parte de la visión artificial, hace un gran aporte en área de la salud para el análisis de imágenes médicas para la detección de cáncer, tumores y otro tipo de enfermedades. A raíz del surgimiento de Covid-19 se ha aplicado la segmentación semántica basada en redes neuronales convoluciones como método de detección del virus, esto se realiza a través del análisis de la radiografía de tórax. El hecho de aplicar segmentación semántica en la detección de Covid-19 y en el entorno médico en general permite no solo reducir el riesgo de contagio de Covid-19 del personal de la salud que se encarga de dar diagnóstico, sino también aporta soluciones de alta precisión en el diagnóstico de enfermedades acelerando el tratamiento de los pacientes. [6].

En ese sentido, la segmentación semántica también contribuye ampliamente en el campo de la conducción autónoma, ya que al aplicar un modelo de aprendizaje profundo logra segmentar de una escena clases importantes como lo pueden ser los peatones, automóviles y ciclistas, permitiendo una conducción más segura y confiable [7]. Así como también puede contribuir en el reconocimiento de escritura, ya que es capaz de extraer palabras y líneas de documentos escritos a mano, permitiendo la digitalización de documentos [8], de esta manera se reduce el tiempo y el espacio de almacenamiento, además de contribuir a la accesibilidad de información y a la preservación del medio ambiente.

Por otro lado, la segmentación semántica también tiene aplicación en el reconocimiento de objetos, por ejemplo en el caso de pruebas de rayos X para el control de equipaje de un aeropuerto, debido a que por medio de la integración de una red convolucional y un autocodificador adverso logra hacer la extracción de características de una imagen de rayos X permitiendo identificar posibles objetos peligrosos como armas y explosivos [9]. Aplicar segmentación semántica en este entorno aumenta la confianza de los pasajeros al reducir posibles problemas de seguridad y demuestra factibilidad del uso de la segmentación semántica como una herramienta para diferentes situaciones sociales.

Finalmente, la segmentación semántica ha demostrado ser extremadamente útil en el mundo actual, ya que ha abierto nuevas oportunidades y beneficiado a las personas con los últimos

avances tecnológicos. Un ejemplo de esto se observa en el campo de la medicina, donde se han logrado resultados más precisos en las pruebas, lo que ha permitido disfrutar de mejores ofertas de atención médica. Del mismo modo, la segmentación semántica ha contribuido en el aumento de la percepción de seguridad vial en las personas, debido a que los sistemas de automóviles ahora pueden identificar las señales de tráfico e incluso navegar procesos complicados relacionados con ir de un lugar a otro.

Este proyecto se enmarca dentro del área de Visión Artificial e Inteligencia Computacional aplicadas a la navegación y control de robots autónomos. Algunos de los proyectos que se enmarcan dentro de esta área incluyen control basado en técnicas de aprendizaje profundo [10], [11], [12], optimización basada en enjambres de partículas [13] y lógica difusa [14]. Además de los proyectos que utilizarán el sistema de segmentación semántica y que se nombran en el trabajo futuro de este proyecto.

Capítulo 2

Estado del Arte

La segmentación semántica es un algoritmo que asocia una etiqueta o categoría a cada pixel presente en una imagen [15]. Con respecto al campo de la visión por computadora como parte esencial de la inteligencia artificial, la segmentación semántica permite mejorar la capacidad de interpretar y comprender el mundo real. Lo anterior se debe a que, al aplicar imágenes digitales de cámaras, videos y modelos de aprendizaje profundo, las máquinas pueden reconocer y clasificar objetos y luego reaccionar a lo que "ven". Es así como la segmentación semántica se convierte en una herramienta para simplificar e incorporar técnicas de aprendizaje profundo en conceptos como inteligencia artificial y aprendizaje automático. Esto ha permitido que los principales retos sean en el área de la visión por computador para realizar con mayor precisión las diferentes tareas de clasificar, localizar y etiquetar semánticamente los elementos que contiene una imagen para interpretar el contexto de esta, a esta tarea se le conoce como reconocimiento de la escena [16][17].

Los primeros trabajos de segmentación de imágenes se remontan a 1965 con el trabajo planteado por Robert W. Floyd, en el que analiza y diseña un algoritmo para identificar los bordes entre diferentes partes de una imagen [18]. El trabajo desarrollado por Floyd se compone de dos partes, la semántica que se refiere al significado y la sintaxis que se refiere a la forma [19].

A partir de entonces, el campo de la segmentación de imágenes ha presentado importantes avances; en los primeros años del siglo XXI se plantearon y diseñaron varios métodos en el procesamiento de imágenes digitales para resolver esta tarea, como lo son: la segmentación de umbral, la segmentación de región, la segmentación de borde, características de textura, agrupamiento, entre otros [20][21]. El desarrollo de estos métodos se realizó con el fin de construir clasificadores; todos los avances son una larga acumulación de procesos tecnológicos para

afrontar ciertas dificultades como asignar una determinada etiqueta o categoría de cada uno de los píxeles de una imagen.

Desde el 2000 hasta el 2010, se plantearon cuatro métodos principales: teoría de grafos, agrupamiento, clasificación y combinación de agrupamientos [21]. Específicamente, en el campo de segmentación semántica, trabajos como el de Xuming [22], Shotton [23] y Torralba [24] permiten dar una perspectiva general de las técnicas y enfoques utilizados para llevar a cabo esta tarea y de esta manera incluir características contextuales o aprender modelos discriminativos de clases de objetos en el etiquetado de imágenes. Sin embargo, a pesar de la cantidad de técnicas tradicionales de procesamiento de imágenes, el surgimiento de las redes neuronales y el aprendizaje profundo impulsaron significativamente los desarrollos en segmentación semántica. En el año 2010, surge *Convolutional Neural Network* (CNN o red neuronal convolucional) como una herramienta de procesamiento visual muy eficiente debido a su capacidad de aprender características jerárquicas y clasificar toda la imagen en una de muchas categorías predefinidas [25]. Posteriormente, debido al trabajo realizado por Alex Krizhevsky con *ImagenNet* en el área de clasificación de imágenes, se empezaron a plantear enfoques basados en aprendizaje profundo, siendo actualmente una tendencia en el desarrollo de trabajos de segmentación semántica [26][27].

En los últimos años, la segmentación se ha convertido en un problema básico en el campo de la visión por computador y el aprendizaje automático, pues se considera que solo encontrar las etiquetas de clase contenidas está lejos de ser suficiente. Las etiquetas de clase contenidas deben ser variadas dependiendo del contexto; por ejemplo, en espacios cerrados las etiquetas pueden estar clasificadas en los diferentes objetos que componen la escena. Otro ejemplo en un campo de acción diferente al propuesto en este anteproyecto son los vehículos autónomos, las etiquetas de píxeles pueden ser humanos, carreteras, automóviles, etc. [25].

En este punto es importante considerar que la mayoría de los trabajos de comprensión de escena hace una década no tenían en cuenta las interacciones físicas en el entorno. Es por eso que Hoiem en el artículo *Indoor Segmentation and Support Inference from RGBD Images* plantea uno de los primeros enfoques con redes neuronales con el fin de interpretar las superficies principales, objetos y relaciones de apoyo de una escena interior a partir de una imagen RGB-D. En primer lugar, se infirió en la estructura general en 3D de la escena para luego analizar conjuntamente la imagen en objetos separados y estimar sus relaciones de apoyo. Para esto se plantea un conjunto de datos de 1449 imágenes RGB-D y se realiza un etiquetado denso por píxel usando *Amazon Mechanical Turk*. Además, se realiza el modelado de la estructura de escenas interiores calculando las normales de superficie, luego se encuentra las direcciones de escena, después

se propone planos 3D usando RANSAC en los puntos 3D. Posteriormente, se realiza la segmentación de la imagen en regiones que corresponden a objetos o superficies haciendo uso del código y del algoritmo de Hoiem. De esta manera, las señales 3D pueden informar mejor una interpretación 3D estructurada [28].

A partir de entonces, los resultados de detección de objetos y la segmentación semántica en el campo de visión han mejorado rápidamente. Han sido muchas las aplicaciones en las que se utilizaron funciones de CNN para reconocer objetos en una imagen; pero, en algunos casos, esta ubicación solo se hizo con cuadros delimitadores y no a nivel de pixel. Este problema fue investigado por Kaiming He y el equipo de investigación de Facebook en el trabajo *Mask R-CNN*. El artículo presenta un enfoque que detecta objetos e instancias en una imagen mediante el diseño de una red neuronal profunda. El método se encuentra basado en *Faster R-CNN* que realiza la detección de objetos y a la cual se le agrega una rama que genera una máscara binaria para el reconocimiento por pixel. Además, se realiza una modificación sobre *RoIPool* para que se alinee con mayor precisión utilizando una capa libre de cuantización conocida como *RoIAlign* [29].
Siguiendo un enfoque similar al expuesto anteriormente, en el estudio presentado en [30] , se lleva a cabo la segmentación semántica de caminos y pasillos con el propósito de desarrollar sistemas de asistencia para individuos con discapacidad visual. Este proyecto implementa la utilización de R-CNN junto con el concepto de superpíxeles para lograr la identificación precisa de caminos y pasillos, tanto en entornos interiores como exteriores. Además, se realizan experimentos que consideran variaciones en la iluminación, evaluando el rendimiento del algoritmo en diferentes momentos del día y ante exposición a la luz solar y artificial.
Otro uso bastante interesante es el propuesto en [31], donde se propone un algoritmo distribuido de detección de víctimas mediante segmentación semántica utilizando redes neuronales convolucionales (R-CNN) en cuadricópteros para operaciones de búsqueda y rescate. Cuando un cuadricóptero identifica a una víctima potencial, hace que los cuadricópteros vecinos formen un subenjambre alrededor de la víctima, validando su estado. Los algoritmos de Consenso de Estimación (EC) y Consenso de Estimación Máxima (M-EC) basados en CNN se utilizan para acordar la detección de víctimas, lo que demuestra la eficacia de CNN en situaciones de visibilidad limitada y condiciones inciertas causadas por fuego y humo.

Un ejemplo de segmentación semántica aplicada se encuentra en la identificación de escenas al interior de un espacio cerrado. Sin embargo, entre los principales problemas de la segmentación semántica con RGB-D en este tipo de áreas está la tarea de extraer y fusionar eficazmente las características de profundidad junto a las de color. En el trabajo de Lee con el fin de explorar en la capacidad que tiene una red neuronal para integrar la información de profundidad, se

presenta una red que extiende la idea central del aprendizaje residual a la segmentación semántica RGB-D. La red captura las características RGB-D CNN multinivel incluyendo bloques de fusión de características multimodales y bloques de refinamiento de características multinivel. Donde, los bloques de fusión de características aprenden las características residuales RGB y de profundidad y sus combinaciones. Los bloques de refinamiento de características aprenden la combinación de características fusionadas. En esta arquitectura, se pueden entrenar y fusionar eficazmente las características de profundidad y RGB de varios niveles, al tiempo que se conserva la ventaja clave de la conexión de salto, es decir, todos los gradientes fluyen efectivamente hacia atrás a través de las conexiones residuales a las características de entrada de *ResNet* [32]. Por otra parte, con el mismo objetivo, Keuper propone una red neuronal convolucional multivista mediante una capa de agrupación espacio temporal de datos (STD2P) basada en superpíxeles para la segmentación semántica con información sustancialmente rica [33]. De esta manera, las redes propuestas producen una segmentación de alta calidad de una sola imagen, aprovechando las características y la información de vistas adicionales de la misma escena.

En los sistemas de recomendación se ha utilizado la segmentación semántica para la detección y clasificación de diferentes tipos de prendas de vestir, como muestran Reyes et al. En [34] se diseña un sistema de recomendación de estilos de vestimenta basado en una base de datos inicial y con un proceso de adaptación del cliente. El sistema aprende de los diferentes estilos del usuario y recomienda automáticamente combinaciones de prendas de vestir después de segmentar las diferentes opciones existentes del guardarropa. Este sistema es altamente adaptable y preciso de acuerdo con los resultados experimentales del trabajo de investigación.
Por otro lado, entre las aplicaciones de la segmentación semántica también se encuentra el campo de la conducción autónoma, un ejemplo de esto se evidencia en el artículo *Importance-Aware Semantic Segmentation for Autonomous Vehicles*. En este proyecto se tiene en cuenta que el reconocimiento de algunos objetos es más importante que otros, por lo cual no todos los métodos de segmentación semántica pueden ser aplicables en el desarrollo de sistemas de conducción autónoma. Los métodos para estas tareas deben tener en cuenta diferentes clases de objetos según su nivel de importancia para una conducción segura. A raíz de lo anterior, el artículo reemplaza la función de pérdida por entropía cruzada por una función de pérdida basada en estructura jerárquica que pone más énfasis en segmentar con precisión los objetos importantes que los menos importantes. Además, comprueba el funcionamiento de esta nueva función de pérdida en diferentes redes neuronales profundas como *Segnet*, FCN y ENet [7]. Del mismo modo, en otros proyectos de vehículos autónomos como el propuesto por Castro y Roncancio también se usa como base modelos de redes neuronales como *ResNet* y se destaca el buen desempeño en circunstancias limitadas generado por la disminución de épocas junto al robustecimiento de la

red [35].

Otro ejemplo de aplicación de segmentación semántica para el desarrollo de sistemas de navegación es el proyecto *Semantic RGB-D SLAM for Rescue Robot*. En el artículo se plantea un modelo semántico para realizar localización y mapeo simultáneo (SLAM) en robots de rescate en terrenos desafiantes. Para esto se divide el trabajo en tres partes. En primer lugar, se desarrolla el sistema RGB-D SLAM para obtener un mapa denso de nubes de puntos y la posición precisa del robot. En segundo lugar, se extrae etiquetas semánticas de imágenes RGB-D mediante aprendizaje supervisado, se entrena una red neuronal convolucional basada en arquitectura Inception-v3 porque ofrece mayor rendimiento a la segmentación por eficiencia computacional y extracción de características. En tercer lugar, se filtra las etiquetas semánticas y mapas, ya que los resultados dados por la CNN pueden contener etiquetas dispersas de clases incorrectas. Para filtrar las etiquetas se realiza una operación de "erosión" y posteriormente una operación de "relleno de inundación". Finalmente, se realiza la planificación de la ruta por medio de la aplicación del método conjunto de niveles de marcha rápida (FFM) [36].

Si bien, la segmentación semántica se usa ampliamente en el establecimiento de mapas semánticos, un buen modelo de segmentación semántica todavía tiene la desventaja de un rendimiento insuficiente en tiempo real para aplicaciones de conducción autónoma. Según esto, en el artículo *Research on the Application of Semantic Segmentation of driverless vehicles in Park Scene* se propone un modelo de red PFPN (Panoptic Feature Pyramid Network) mejorado para reducir el tiempo de segmentación semántica. En este algoritmo, se recorta la función de rama de segmentación de instancia en el PFPN original y se corta la pequeña capa de entidad de destino extraída de la rama de segmentación semántica, a fin de reducir el tiempo de segmentación del modelo. Además, para verificar la exactitud y viabilidad del modelo, se establece un entorno de prueba en la escena del campus. En este tipo de trabajos se evidencian los avances generados en el campo de segmentación semántica, por ejemplo, en este caso se logran mejoras en los resultados de segmentación y se reduce el tiempo promedio de segmentación de imágenes a un 22 % [25].

Los anteriores ejemplos, a pesar de ser aplicaciones relacionadas con el campo de la navegación, sirven como referencia del trabajo presente y demuestran que la segmentación semántica en conjunto con las redes neuronales convolucionales ha permitido una gran evolución en el campo de reconocimiento de objetos y la clasificación de escenas. La aplicación de CNN ha alcanzado grandes logros y han utilizado en la clasificación y detección de imágenes; sin embargo, también presentan algunas desventajas en cuestión de almacenamiento, eficiencia de cálculo, entre otros aspectos. Es por esto que en los últimos años se propuso las redes totalmente convolucionales (*Full Convolucional Network* o FCN), ya que no tiene problemas de tamaño de

imágenes y son más eficientes con respecto al almacenamiento y la aplicación de procesos de convolución. En la FCN la clasificación de la imagen parte desde la clasificación a nivel de píxel y esto se genera a partir de la sustitución de todas las capas totalmente conectadas por las capas convolucionales. Según lo planteado por Fude Cao y Qinghai Bao en el artículo *"A Survey On Image Semantic Segmentation Methods With Convolutional Neural Network"*, la FCN sustituye las capas intermedias de convolución, cambiando ligeramente la salida de cualquier número de resultados de clasificación. En este artículo FCN utiliza la red neuronal de clasificación VGG-16 y el entrenamiento se realiza con conjuntos de datos de entrenamiento PASCAL VOC. Por otra parte, para restaurar el mapa de características clasificadas al tamaño original se puede aplicar deconvolución o interpolación bilineal. Según lo anterior se plantea SegNet, la cual es una arquitectura de segmentación semántica, la cual se encuentra basada en FCN. Cao y Qinghai formulan que SegNet puede lograr un entrenamiento de principio a fin, haciendo que sea más eficiente que FCN con respecto a tiempo de entrenamiento y memoria, además de lograr mayor comprensión de la escena [37].

En otros proyectos de segmentación semántica se encuentra que actualmente existe un tema de investigación activo, en donde la segmentación de imágenes ha alcanzado una precisión gracias al gran progreso de las redes neuronales convolucionales y varios conjuntos de datos. Sin embargo, muchas de las aplicaciones del mundo real tienen fuertes demandas por la segmentación de video [38].

En los últimos años, se ha dispuesto de métodos en la propagación de características para la segmentación semántica de imágenes y video, entre estas se encuentran DFF (*Deep feature flow*), NETWarp y DVSNet. En el artículo *Video Semantic Segmentation With Distortion-Aware Feature Correction* profundizan en el uso de estos métodos los cuales se encargan de calcular el flujo óptico entre el cuadro clave y el cuadro actual, y luego reproducir las características del cuadro actual al propagar las características bajo la guía del flujo óptico. En este desarrollo, la interpolación bilineal generalmente se usa como el operador de deformación de características. Los métodos de estimación de flujo óptico basados en las redes neuronales convolucionales son los predispuestos, ya que son fáciles de manejar e integrar en el marco de segmentación de video para el entretenimiento. A pesar del gran progreso en las últimas décadas, la estimación del flujo óptico sigue siendo un problema desafiante. En particular, la obstrucción causada por el movimiento de la escena hace que la estimación del flujo óptico sea incorrecta, ya que no existe una correspondencia visual por los píxeles ocluidos. Esto permite que cuando el flujo óptico

impreciso se usa en la propagación de características, las características producidas se distorsionan y puedan generar aún más resultados de segmentación incorrectos. Por este motivo, algunos métodos existentes que pueden aliviar la distorsión de las características al modular son: DFF que adjunta un campo de escala, la estimación de flujo óptico y ajusta las características propagadas a través de la multiplicación por elementos. Accel propone extraer características del marco actual con un modelo ligero y luego fusionar las extraídas [38].

El desarrollo de trabajos antes mencionados como SLAM RGB-D semántico para rescate o *Rdfnet: Rgb-d multi-level residual feature fusion for indoor semantic segmentation* entre otros, puede presentar una guía para llevar a cabo la segmentación semántica como una aplicación para problemas reales de navegación de robots móviles ya que fusiona etiquetas semánticas obtenidas de la segmentación semántica y mapas generados. Sin embargo, es claro que las aplicaciones de segmentación semántica no solo se reducen al campo de la robótica móvil o sistemas físicos en general, un ejemplo de esto es el proyecto denominado como *Image style transfer algorithm based on semantic segmentation*. Este proyecto intenta resolver el problema de la falta de coincidencia semántica en el proceso de transferencia de estilo de imagen. Según esto, a partir de una red de segmentación semántica basada en una máscara R-CNN, se extrae información semántica para obtener una imagen de máscara con la misma resolución que la imagen de entrada y con cada píxel etiquetado según la categoría correspondiente. Posteriormente, se realiza la transferencia de estilo por medio de la extracción de las características de contenido y las características de estilo de la imagen de contenido. Estas características son ingresadas en una red compuesta basada en redes neuronales muy profundas (*Very Deep Convolutional Networks* o VGGNET) para obtener una nueva imagen que combine las características de contenido y las características de estilo. En este punto, hay que considerar que cada elemento de característica de contenido y de estilo corresponde al mapa de características de diferentes capas de la red neuronal convolucional. Finalmente, se implementa una red de ajustes de estilos, para esto se introduce la información semántica en la granularidad del bloque de imagen. Después, los bloques de imagen se combinan en función de las características de estilo y la información semántica. De esta manera se logra realizar la transferencia de estilo a nivel semántico [39].

Según lo comentado en este capítulo, es claro que mediante la segmentación semántica se puede reconocer diferentes objetos en una imagen RGB según color y textura, permitiendo facilitar su evaluación. Recientemente, los investigadores pudieron realizar la segmentación semántica correctamente en imágenes, sin embargo, los métodos basados en imágenes RGB carecen de suficiente información para realizar la segmentación semántica de escenas complejas. El trabajo *A Survey on Indoor RGB-D Semantic Segmentation: from Hand-Crafted Features to Deep Convolutional Neural Networks* abarca el problema de la segmentación que ocurre debido a la amplia gama

de relaciones contextuales entre objetos, los cambios de iluminación y las variaciones de apariencia, escala, pose y punto de vista. Adicionalmente, plantea que la segmentación semántica RGB-D con información de profundidad logra mejores resultados de segmentación mediante muchos experimentos; sin embargo, falta una encuesta exhaustiva sobre el tema. En definitiva, el artículo ofrece una revisión detallada de la segmentación semántica RGB-D de acuerdo con el progreso de la investigación en los últimos años. Además, a partir de ahí evidencia que redes como SegNet son útiles para aplicaciones con poca memoria disponible y, por el contrario, FCN requiere mucha más memoria y tiene un alto coste computacional [40].

La segmentación semántica en interiores sigue siendo uno de los problemas más desafiantes no solo por el análisis de escenas complicadas, sino también debido a los objetos complejos y variados con oclusiones severas. Se han logrado avances significativos en el diseño de CNN para la segmentación semántica RGB, sin embargo, estas CNN no se adaptan ampliamente para la segmentación RGB-D debido a la asimetría entre las modalidades RGB y de profundidad. En el artículo *RGBxD Learning depth-weighted RGB patches for RGB-D indoor semantic segmentation* se propone una forma novedosa de aprender la fusión de RGB e información de profundidad en una etapa temprana. El método propuesto trata la información RGB y de profundidad de forma asimétrica, y es el primer enfoque que aprende a fusionarlos de forma multiplicativa para la segmentación RGB-D. Esta propuesta enriquece la discriminación de los parches de imagen y hace que la convolución sea más consciente de la geometría en el aprendizaje. Los extensos experimentos y estudios de ablación en los desafiantes puntos de referencia de segmentación semántica NYUDv2, SUN RGB-D y Cityscapes muestran que el RGB D propuesto ofrece una mejora constante en varias líneas de base [4].

De la misma forma, el artículo *Detección y segmentación de objetos en imágenes panorámicas* profundiza en el uso de segmentación semántica en el reconocimiento de objetos en escenas de interior, en este caso a través del análisis de imágenes panorámicas. Las imágenes panorámicas brindan un contexto del entorno total, sin embargo, cuando se utiliza este tipo de imágenes en el reconocimiento de objetos, las técnicas convencionales quedan obsoletas debido a distorsiones en la imagen. Para llevar a cabo el análisis de este tipo de imágenes, el artículo propone el desarrollo de una red neuronal convolucional. La arquitectura de la red tiene estructura *encoder-decoder* con un extractor de características. Esta red está basada en BlitzNet, pero con algunas modificaciones que le hacen posible trabajar con la distorsión que introducen las imágenes panorámicas. En principio se tiene una imagen RGB de entrada la cual es procesada por un extractor de características, en este caso es una red neuronal convolucional conocida como ResNet50. Después de extraer las características se obtiene una imagen cuyo tamaño se ha reducido con respecto al de la entrada, por lo que para realizar la segmentación semántica se aumenta la resolución de

la imagen. Finalmente, los mapas de características obtenidos se introducen como entrada de las ramas de detección y de segmentación [41].

A pesar de los avances y los diferentes proyectos realizados en segmentación semántica, se ha comprobado que aún hay dificultad para la localización precisa de los límites de las etiquetas de clase a nivel pixel de imágenes. De acuerdo con eso, la localización precisa de la segmentación semántica está atrayendo cada vez más la atención de los métodos basados en el aprendizaje profundo, estos dominan las actuaciones destacadas, especialmente de las redes neuronales convolucionales profundas (DCNN). Sin embargo, en las DCNN las salidas no están lo suficientemente localizadas como para establecer límites de objetos precisos debido a sus propiedades de invariancia, lo que hace que la recuperación de la segmentación semántica sea un problema. Teniendo en cuenta esto, el documento *A survey on deep learning-based precise boundary recovery of semantic segmentation for images and point clouds* realiza un estudio de la recuperación de límites precisos para la segmentación semántica, centrándose principalmente en imágenes 2D y nubes de puntos 3D. Las técnicas se examinan desde dos perspectivas: las estructuras de modelos y los tipos de datos. Para esto se dividen las técnicas de recuperación en cuatro categorías (predicción multiescalar, representación de superpixels, campos aleatorios condicionales y alternativas). A partir de la comparación y el análisis de las CNN en 2D, el documento demostró que es difícil realizar mayores avances sobre el propio algoritmo, pero aún hay muchas posibilidades al explorar métodos híbridos [42].

Con el método de segmentación semántica junto con redes neuronales, se debe considerar que la mayoría de las técnicas de aprendizaje automático deben satisfacer que el conjunto de entrenamiento y prueba son independientes y están distribuidos de manera idéntica. Ahora bien, es difícil satisfacer esta suposición en la aplicación práctica. El artículo *Review the state of the art technologies of semantic segmentation based on deep learning* se enfoca en diferentes propuestas de trabajos de segmentación, ya que es una técnica que tiene infinidades de aplicaciones, además habla del rendimiento en tiempo real del modelo que debe tenerse en cuenta en la aplicación práctica. También analiza los factores claves que afectan el rendimiento en tiempo real y finalmente, este documento resume los desafíos y las direcciones de investigación prometedoras de las tareas de segmentación semántica basadas en el aprendizaje profundo [43].

Capítulo 3

Marco Teórico

3.1. Segmentación semántica

La segmentación semántica es un algoritmo de aprendizaje profundo capaz de asociar una etiqueta o categoría con cada píxel presente en una imagen [27], procesando así las imágenes de manera detallada y precisa. Esta técnica es una tarea avanzada que contribuye a la comprensión integral de una escena, lo que la hace importante en visión artificial, ya que cada vez más aplicaciones infieren conocimiento a partir de imágenes. Las aplicaciones van desde la interpretación de varios tipos de escenarios hasta la realización de tareas como la conducción autónoma, pasando por el análisis médico para facilitar el diagnóstico y tratamiento de enfermedades [44].

Al igual que con otros problemas y tareas relacionados con la visión artificial, en los últimos años se han propuesto y demostrado grandes avances en los modelos de segmentación semántica, en gran parte gracias al reciente auge de los enfoques basados en el aprendizaje profundo [45]. En este sentido, la segmentación semántica se logra utilizando arquitecturas profundas, generalmente redes neuronales convolucionales, que superan ampliamente a otros métodos en cuanto a precisión y, en ocasiones, incluso en eficiencia [44][45]. Con lo anterior en mente, es importante comprender cómo funciona el etiquetado de píxeles mediante la aplicación de modelos de segmentación semántica.

3.2. Modelo de segmentación

Los modelos de segmentación semántica proporcionan, como salida, mapas de segmentos según la imagen de entrada. Cada mapa de segmentos tiene n canales, donde n corresponde a la cantidad de clases que es capaz de segmentar. El propósito de estos modelos es tomar una imagen y generar un mapa de salida en la cual el valor del píxel de la imagen de entrada se transforma en un valor de etiqueta de clase. Por ejemplo, una imagen de color RGB cuyas dimensiones son (alto x ancho x 3) genera un mapa de segmentación donde cada píxel contiene una etiqueta de clase representada como un número entero n (alto x ancho x 1) como se representa en la figura 1. En este sentido, en la figura se tiene una imagen de entrada de la cual se obtiene un mapa de segmentación simplificado de etiquetas de clase de píxeles [46].

FIGURA 1: Ejemplo de máscara de segmentación [46]

Como se ha mencionado antes, para obtener el mapa de segmentación es necesario crear un canal de salida para cada una de las clases posibles. En el caso del ejemplo se tienen 5 clases diferentes, las cuales corresponden a persona, cartera, vegetación, acera y edificio [46]. La figura 2 muestra los mapas para cada clase.

3.3. Red Neuronal

Una red neuronal es un modelo computacional que emula el funcionamiento del cerebro humano. En esencia, una red neuronal es un conjunto de algoritmos que son capaces de procesar y analizar datos con el fin de aprender patrones y hacer predicciones [47].

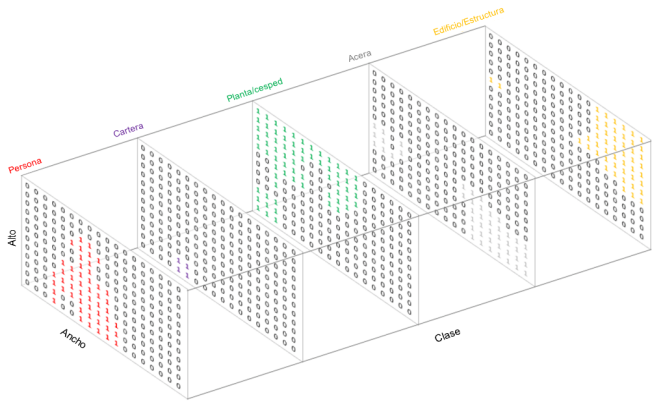

FIGURA 2: Mapa de segmentación [46]

3.3.1. Redes Neuronales Convolucionales

Las redes neuronales convolucionales son algoritmos de aprendizaje profundo que pueden entrenar grandes conjuntos de datos con millones de parámetros, en forma de imágenes 2D como entrada y convolucionarlos con filtros para producir los resultados deseados [48]. Las redes neuronales se encuentran inspiradas biológicamente, por tanto, también están compuestas por neuronas. Las neuronas en este caso corresponden a una función matemática que toma varios valores como entradas y a cada una de estas entradas se les asigna un vector de pesos que hacen que cada neurona sea única. Cada neurona trabaja en su propio campo y a su vez se encuentra conectada a otras neuronas para de esta manera lograr cubrir el campo de trabajo total [49].

Para comprender la red neuronal convolucional de manera original, es esencial entender que la convolución es una operación matemática que implica la aplicación de una función a otra función, generando un resultado que puede considerarse como una combinación de ambas. Las convoluciones resultan útiles para detectar estructuras simples en una imagen, y al combinar múltiples convoluciones, se pueden construir funciones más complejas. En una red convolucional, este proceso se lleva a cabo en varias capas, donde cada una realiza una convolución sobre el resultado de la capa anterior [50].

La arquitectura de una red neuronal convolucional consta de varias capas diseñadas para procesar los datos de entrada en forma de imágenes o videos. Las capas más comunes de las CNN son:

- Capas convolucionales: Es el bloque de construcción central de una CNN, y tiene como

objetivo extraer características propias de cada imagen. Requiere de algunos componentes, que son datos de entrada, un filtro y un mapa de características. Además, se utiliza un detector de características, también conocido como *kernel* o filtro, que se desplaza a través de los campos receptivos de la imagen para verificar la presencia de características [50].

- Capa de *Pooling*: Su función principal es la de reducir la dimensión espacial de la salida de las capas convolucionales. Para lograr esto, se divide la imagen en regiones y se aplica una función de reducción en cada una de ellas. Algunas de las funciones de reducción más comunes son el promedio y el máximo, que calculan los valores de píxeles en cada región [50].
- Capa completamente conectada: Es aquella en la que cada nodo de la capa de salida se conecta directamente a un nodo de la capa anterior. Esta capa desempeña la tarea de clasificación, utilizando las características extraídas a través de las capas anteriores y sus diversos filtros para determinar la probabilidad de que la imagen pertenezca a cada una de las categorías posibles [50].

3.3.2. Entrenamiento red neuronal

Las redes neuronales convolucionales aprenden a reconocer una diversidad de objetos dentro de imágenes, para esto es necesario realizar un entrenamiento previo a partir de datos que incluyen características específicas de la tarea [49]. Este proceso de entrenamiento se realiza con el fin de que las neuronas de la red puedan captar características únicas de un elemento y a la vez la tarea pueda ser generalizada. Para realizar la clasificación deseada es necesario especificar una superficie de decisión por medio del valor de peso y del valor de umbral. En términos generales, el ajuste de los pesos y umbrales en una red suele realizarse a través de un proceso iterativo de presentación repetida de ejemplos de la tarea requerida. Durante cada presentación, se realizan pequeños cambios en los pesos y umbrales para alinearlos mejor con los valores deseados, por tanto, la precisión de una red será consecuencia del ajuste de pesos en términos de la función de pérdida [51].

En este punto, es evidente la importancia del uso de una red neuronal en el desarrollo de la segmentación semántica; sin embargo, el hecho de entrenar una red neuronal desde cero puede llegar a ser una tarea complicada. Normalmente, la mayoría de los modelos utilizados en este campo necesitan muchos recursos y tiempo de ejecución significativo, además que los conjuntos de datos contienen millones de muestras, lo que no hace factible construir una amplia colección de datos para las diferentes tareas de inteligencia artificial.

3.3.3. Transfer Learning

El aprendizaje por transferencia (*Transfer Learning*) es un método de aprendizaje automático, el cual se caracteriza por utilizar un modelo previamente entrenado como base para el desarrollo de otras tareas. Esta técnica resulta efectiva en el proceso de aprendizaje, ya que es posible ajustar en conjuntos de datos más pequeños un modelo que ya ha sido entrenado por un conjunto de datos grande, evitando tener que entrenar un nuevo modelo desde cero. De esta forma, se aprovecha el conocimiento previamente adquirido por el modelo base, lo que puede mejorar significativamente la eficiencia y precisión del proceso de aprendizaje. Además de ahorrar tiempo de capacitación y recursos informáticos [52] .

Los enfoques de aprendizaje de transferencia se dividen comúnmente en dos fases principales: la fase de preentrenamiento y la fase de ajuste fino. En términos generales, las redes neuronales intentan detectar ciertas características en función de la capa en la que se encuentran. Por ejemplo, en las primeras capas buscan detectar bordes, en las capas intermedias detectan formas y en las capas posteriores se detectan características más específicas. Cuando se usa *Transfer Learning*, se pueden reutilizar las capas iniciales e intermedias, de tal manera que solo es necesario entrenar las últimas capas según la tarea a desarrollar. Esto permite aprovechar los datos etiquetados en las primeras capas, por su parte, las capas finales de la red neuronal se eliminan y se reemplazan con nuevas capas que se entrenan con un conjunto de datos más pequeño y específico para la tarea a desarrollar [52] .

3.4. Red de aprendizaje profundo: YOLO

YOLO (*You Only Look Once*) es un algoritmo en el área de la visión artificial, diseñado específicamente para la detección de objetos. A diferencia de otras arquitecturas, YOLO tiene la capacidad de realizar predicciones simultáneas múltiples cuadros delimitadores y probabilidades de clase en una sola ejecución, lo que le confiere una ventaja en términos de velocidad. El algoritmo se fundamenta en una red neuronal convolucional (CNN) que utiliza características de imagen para generar cuadros delimitadores, permitiendo que la red identifique los objetos presentes en la imagen. La imagen se divide en una cuadrícula de tamaño SxS, y cada celda de la cuadrícula se encarga de detectar cualquier objeto presente en esa región específica [53][54].

Cada celda predice N cuadros delimitadores junto con sus puntuaciones de confianza. Estas puntuaciones indican la fiabilidad del modelo y que tan preciso es la predicción de un objeto. Lo anteriormente descrito se representa en la ecuación 3.1 [54].

$$Pr(Objeto) * IOU \tag{3.1}$$

Entre ellos, la probabilidad Pr(Objeto) se refiere a la confianza asignada por el modelo a la detección de un objeto en particular. Esta probabilidad se basa en la salida de la red neuronal y puede considerarse como una medida de la certeza que tiene el modelo en su predicción. Por su parte, IOU significa Intersección sobre Unión. Según la ecuación, si no hay ningún objeto en la celda, la puntuación de confianza debería ser cero. En cambio, si hay un objeto en esa celda, la puntuación será igual a Intersección de Unión (IOU) [54].

La arquitectura de YOLO varía según la versión y la aplicación, sin embargo, a nivel general YOLO se divide en dos partes principales. La primera parte es la extracción de características, la cual se encarga de procesar la imagen de entrada y generar un conjunto de características que sean útiles para la detección de objetos. La segunda parte es la detección, la cual utiliza dichas características para predecir la ubicación y clase de los objetos en la imagen. La detección se basa en una combinación de capas convolucionales y capas completamente conectadas [53].

3.4.1. YOLOv7

YOLO cuenta con varias versiones, entre las últimas versiones se encuentra YOLOv7. La arquitectura del modelo mejora a los modelos anteriores en cuestión de velocidad y precisión en la detección de objetos. Además de la identificación de objetos, esta arquitectura también se aplica en la segmentación. YOLOv7 ha demostrado mejores resultados cuando se evalúa su rendimiento con respecto a versiones anteriores de YOLO. A continuación se muestra el rendimiento en cuestión de la cantidad de parámetros, la precisión promedio y la cantidad de operaciones de punto flotante de algunos modelos anteriores de YOLO y de las versiones de YOLOv7 (tabla 1).

3.4.2. Arquitectura Utilizada YOLOv7-Tiny

YOLOv7-Tiny es una versión optimizada y modificada de la arquitectura YOLO, diseñada para la detección de objetos en tiempo real con una eficiencia computacional mejorada[56].

En primer lugar, YOLOv7-tiny usa múltiples redes neuronales convolucionales para extraer características significativas de las imágenes de entrada. Estas capas convolucionales combinadas con capas de activación no lineales son responsables de detectar patrones y características en

Model	Param	FLOPs	Size	APval	APval 50	APval 75	APval S	APval M	APval L
YOLOv4-tiny	6.1	6.9	416	24.9 %	42.1 %	25.7 %	8.7 %	28.4 %	39.2 %
YOLOv7-tiny	6.2	5.8	416	35.2 %	52.8 %	37.3 %	15.7 %	38.0 %	53.4 %
improvement	+2 %	19 %	-	10.3 %	10.7 %	11.6 %	7.0 %	9.6 %	14.2 %
YOLOv4-tiny-31	8.7	5.2	320	30.8 %	47.3 %	32.2 %	10.9 %	31.9 %	51.5 %
YOLOv7-tiny	6.2	3.5	320	30.8 %	47.3 %	32.2 %	10.0 %	31.9 %	52.2 %
improvement	39 %	49 %	-	-	-	-	-0.9	-	0.7

TABLA 1: Comparación de los detectores de objetos de referencia con YOLOv7-tiny [55]

diferentes niveles de abstracción. Cuantas más capas convolucionales se agreguen, mayor será la capacidad de la red para capturar características complejas en los objetos detectados[57][58].

Además de las capas convolucionales, YOLOv7-tiny también utiliza operaciones de concatenación. Estas capas concatenadas combinan características extraídas de diferentes niveles de resolución. Al combinar información de características de alto y bajo nivel, la red puede capturar detalles finos y características gruesas, mejorando la precisión de detección[57][58].

Por otro lado, la arquitectura incluye capas de agrupación máxima (*Max Pooling*) que reducen la resolución espacial de las características extraídas. Este proceso de reducción de la información resulta en una mayor eficiencia de la red al reducir la cantidad de datos que requieren procesamiento posterior. La técnica de agrupamiento máximo ayuda a resumir y retener las características más relevantes mientras reduce la sensibilidad a pequeñas variaciones espaciales en la imagen[57].

A continuación, en la figura 4 se presenta la arquitectura de la red neuronal YOLOv7-tinny.

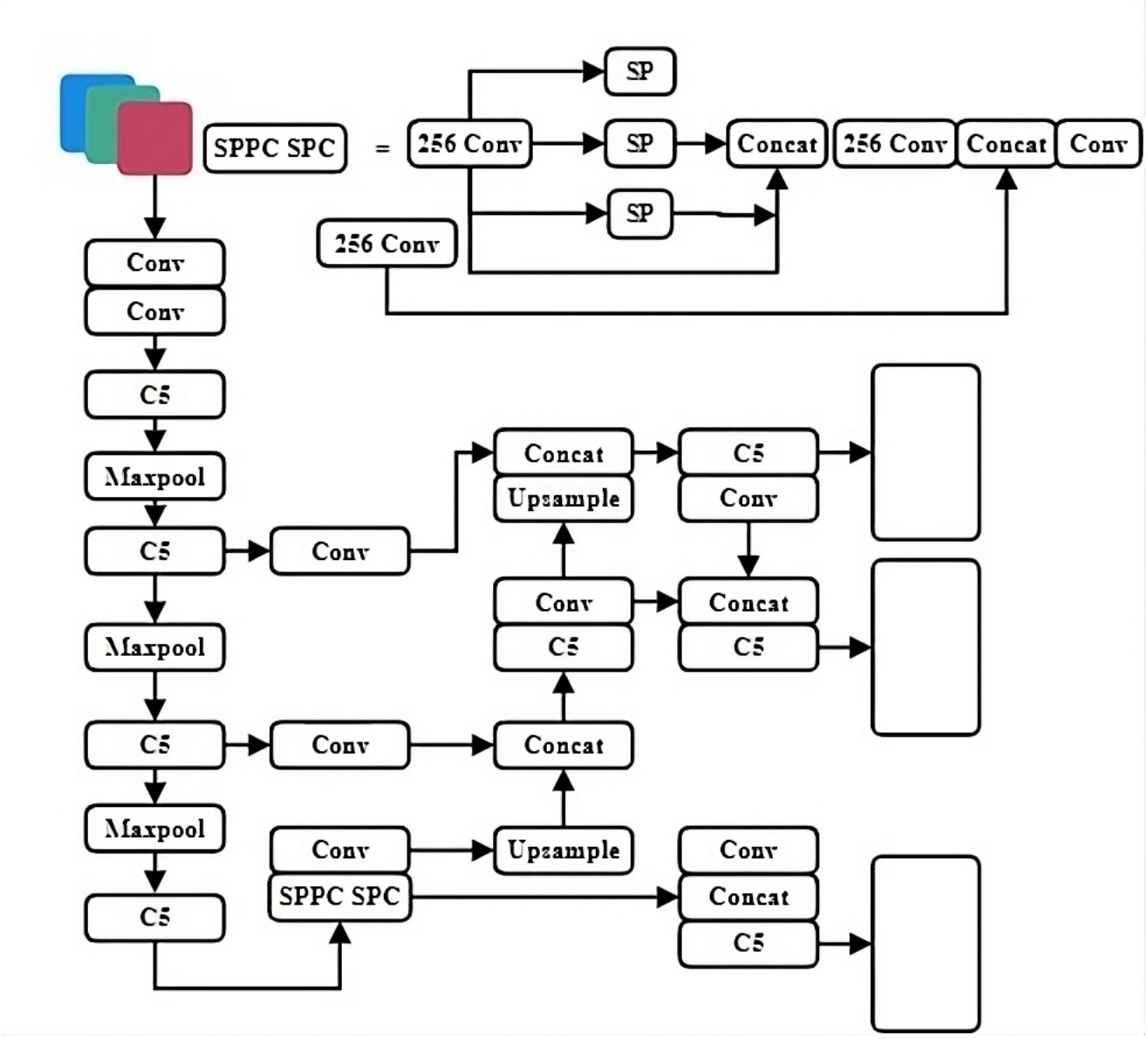

FIGURA 3: Arquitectura de la red neuronal YOLOv7-Tiny

3.5. Arquitectura Mask R-CNN (segmentación semántica)

Mask R-CNN es una extensión del modelo *Faster R-CNN*, que combina la capacidad de detección de objetos con la generación de máscaras de segmentación semántica precisa, su arquitectura consta de varias etapas[59]:

La primera etapa de Mask R-CNN es similar a *Faster R-CNN* y se encarga de la detección de objetos en una imagen. Utiliza una red neuronal convolucional (CNN) para extraer características de la imagen, estas características son fundamentales para identificar los objetos presentes en la escena. A continuación, se utiliza una red de selección de regiones (*Region Proposal Network*, RPN) para generar propuestas de regiones que puedan contener objetos de interés. Estas propuestas se generan en función de las características extraídas previamente[60].

En la segunda parte, se seleccionan las regiones de interés (ROIs) más prometedoras y se clasifican en diferentes categorías utilizando una red de clasificación. Este proceso permite identificar los objetos presentes en la imagen y asignarle una etiqueta correspondiente a su clase. Sin embargo, lo que distingue a Mask R-CNN es su capacidad para generar máscaras de segmentación semántica precisas para cada objeto detectado. Para lograrlo, se agrega una tercera rama a la arquitectura del modelo, conocida como rama de máscara. Esta rama se encarga de generar máscaras binarias que delimitan con precisión los contornos de los objetos en la imagen[60][61].

Las máscaras generadas por la rama de máscara se aplican a las regiones de interés correspondientes en la imagen original, lo que permite una segmentación semántica precisa de los objetos. De esta manera, Mask R-CNN logra la detección y segmentación simultánea de objetos con alta precisión[61].

3.6. Base de datos COCO

El conjunto de datos COCO (*Common Objects in Context*) está diseñado específicamente para abordar desafíos de la detección y segmentación de objetos, proporcionando una colección amplia y diversa de imágenes anotadas[62].

Además, el dataset COCO se ha establecido como un estándar para evaluar el rendimiento de los modelos de visión artificial. Al utilizar esta colección, es posible desarrollar y evaluar algoritmos para realizar tareas como la detección de objetos, la clasificación y la segmentación semántica[62][63].

La base de datos de COCO contiene más de 330.000 imágenes, más de 200.000 de las cuales están anotadas, cubriendo una amplia gama de categorías de objetos (ver figura 4. Este conjunto de datos representa un proyecto colaborativo en el que los principales expertos en visión artificial de instituciones como Google, Caltech y Georgia Tech participan activamente en su mantenimiento y desarrollo[62].

FIGURA 4: Lista de clases de conjuntos de COCO [64]

Capítulo 4

Diseño Metodológico

El presente proyecto busca el desarrollo de un sistema de segmentación semántica para la identificación de objetos específicos. En este sentido, este proyecto se describe como una investigación aplicada, ya que a raíz de la implementación de diferentes algoritmos de segmentación semántica es posible realizar la identificación de puntos de interés. Según lo anterior, el desarrollo del proyecto se divide en cuatro fases.

FASE 1: Recopilación y revisión de la información

En esta primera fase se realizó la recopilación de información acerca de los diferentes tipos de redes neuronales convolucionales, algoritmos de segmentación semántica y métricas de evaluación necesarias para el algoritmo. Se usaron los recursos bibliográficos brindados por la Universidad a través del CRAI (Centro de Recursos para el Aprendizaje y la Investigación), revistas especializadas, trabajos de grado y textos guía.

Para llevar a cabo la recolección de información se ejecutaron una serie de pasos; en primer lugar, se procede a hacer una revisión de trabajos similares al proyecto. En segundo lugar, a partir de los datos obtenidos, se efectúa un análisis de los tipos de redes neuronales en relación con el sistema a desarrollar y los diferentes algoritmos de segmentación semántica.

FASE 2: Definición del algoritmo

Finalizada la etapa de recopilación y revisión de la información, se determinó el algoritmo de segmentación semántica para el desarrollo del sistema a partir de la información consultada.

Para determinar el algoritmo de segmentación semántica se definieron algunos criterios a partir de los requerimientos del sistema como lo son la capacidad de cómputo, el sistema de visión, entre otros.

FASE 3: Implementación del algoritmo

Se realizó la implementación del algoritmo de segmentación semántica y el desarrollo de las redes neuronales en la plataforma computacional elegida, haciendo uso del algoritmo definido mediante Python.

FASE 4: Evaluación de desempeño

En esta etapa final, se realizaron pruebas de funcionamiento del sistema desarrollado, teniendo en cuenta las métricas de evaluación que se definen a raíz de la revisión de documentos. Cabe mencionar que esta etapa de evaluación permite la detección de fallas o ajustes necesarios, que puedan ser corregidos.

Capítulo 5

Desarrollo Conceptual

El desarrollo de este proyecto se divide en dos secciones principales. La primera parte se centra en el hardware, lo que implica el desarrollo estructural del proyecto y la implementación de las conexiones necesarias para su funcionamiento. La segunda parte se enfoca en el desarrollo de software, que consiste en la implementación de algoritmos para el movimiento del motor, la captura de imágenes, la detección de objetos, entre otras funciones.

5.1. Desarrollo Hardware

En cuestión de hadware los dispositivos que se utilizan son:

- NVIDIA Jetson Nano
- Motor Dynamixel AX-12
- Cámara Intel Realsense D435i
- Controlador USB Dynamixel U2D2
- Regulador LM2596 DC-DC
- Batería

La batería de 11.1 voltios y 2200mAh, compuesta por 4 celdas y capaz de realizar una descarga constante de 25C, se encarga de suministrar energía directamente al motor Dynamixel AX-12a. Asimismo, cuenta con un regulador de voltaje de 5 voltios para alimentar la tarjeta Jetson

Nano. La tarjeta Jetson Nano es una placa de computación que opera con el sistema operativo Ubuntu y contiene diversos algoritmos para el funcionamiento del sistema de detección. En esta configuración, la tarjeta Jetson Nano también está conectada a la cámara Intel Realsense D435i, así como al controlador U2D2, que permite el control del motor.

5.1.1. Estructura Mecánica

El diseño de la estructura mecánica se genera teniendo en cuenta las dimensiones y funcionalidad de cada uno de los elementos. Para esta estructura se realiza un diseño en SolidWorks que permita y facilite el movimiento del motor cuando realiza el giro de 360°. La estructura mecánica se encarga de sostener la cámara y proporcionar un espacio para integrar los elementos electrónicos descritos en la sección anterior. El diseño en 3D se puede observar en las figuras 5 y 6

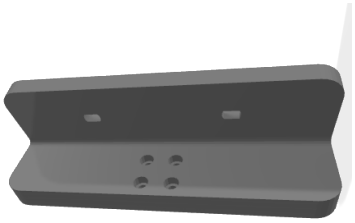

FIGURA 5: Diseño estructura soporte cámara

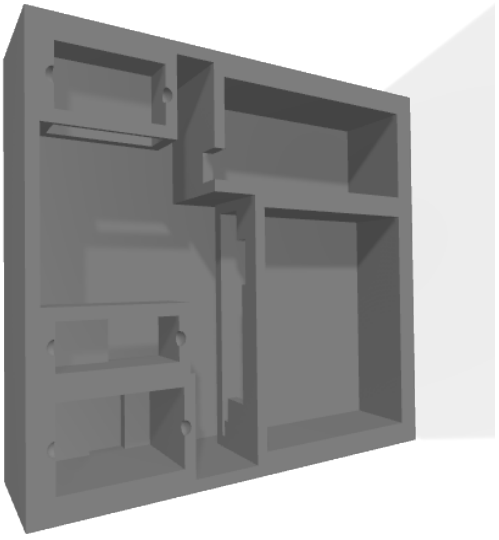

FIGURA 6: Diseño estructura soporte elementos

La estructura se divide en dos partes: una destinada a sostener y permitir el movimiento de la cámara mediante el motor (figura 5), y otra que se encarga de proporcionar un espacio para organizar el resto de los dispositivos (figura 6).

5.2. Desarrollo Software

El software se desarrolla haciendo uso del dispositivo de cómputo Jetson Nano, que ha sido seleccionado por su capacidad de procesamiento, rendimiento y bajo consumo de potencia. La tarjeta Jetson Nano está equipada con un procesador ARM de cuatro núcleos, una GPU NVIDIA, memoria RAM y almacenamiento. Las características del software, junto con las librerías utilizadas, se evidencian en la tabla 6:

Recurso	Especificación
Sistema Operativo	Ubuntu 18.04
Lenguaje de programación	Python 3.6
Librería cámara	pyrealsense2
Librería de visión artificial	OpenCV 4.1.1
Librería cálculos numéricos	Numpy 1.18.5
Librería aprendizaje profundo	Torch 1.12.0
Librería de visualización de datos estadísticos	Seaborn 0.11.0
Librería para interfaces gráficas	Tkinter
Librería para segmentación de estancias	Detectron2

TABLA 2: Especificación del software

La programación del sistema se divide en tres subprocesos independientes, cada uno dedicado a una funcionalidad específica. El primer hilo se encarga del movimiento del motor Dynamixel AX-12a. El segundo implementa el modelo de la red neuronal junto al algoritmo de detección. La tercera tarea se encarga de graficar un radar con las posiciones de los objetos detectados. Adicionalmente, se desarrolla un algoritmo para unir las imágenes tomadas a intervalos regulares de distancia en una imagen panorámica con su respectiva segmentación.

5.2.1. Movimiento del motor

El control y operación del servomotor se realiza por medio del lenguaje Python, utilizando el conversor de comunicación Dynamixel U2D2. Para facilitar esta tarea, se emplean dos librerías. La primera es "dinamixel sdk"que proporciona funcionalidades para el control del motor. La

segunda librería es "msvcrt"que permite importar funciones y excepciones propias de lenguaje C. A partir de este punto, es necesario configurar algunos parámetros del motor desde Python. Estos incluyen la versión del protocolo, la velocidad de transmisión y las direcciones de control específicas del modelo. Las direcciones del motor Dynamixel AX-12a, se detallan en la tabla de referencia 3. Dentro de la biblioteca Dynamixel, se encuentra el método PortHandler que

Recurso	Direcciones
Límite de ángulo en sentido del reloj	6
Límite de ángulo en sentido contrario reloj	8
Habilitar par	24
Posición objetivo	30
Posición actual	36
Versión de protocolo	1
Baudrate	1000000

TABLA 3: Variables de configuración de motor

establece comunicación entre el puerto y el dispositivo Dynamixel. Asimismo, se encuentra el método PacketHandler, encargado de enviar y recibir los paquetes de datos al dispositivo Dynamixel. Estos métodos son utilizados para abrir el puerto de comunicación, configurar la velocidad de transmisión (*Baudrate*) y enviar paquetes de datos al motor Dynamixel para establecer el límite del ángulo en sentido horario y antihorario.

El algoritmo que corresponde al movimiento del motor se desarrolla en un hilo, ya que es necesario garantizar una ejecución simultánea con otros hilos. El motor cambia su posición en intervalos de 30° aproximadamente cada vez que se oprime una tecla. Esto ocurre dentro de un bucle, por tanto, cada vez que se oprime una tecla, se envía la posición objetivo al motor Dynamixel utilizando la función *write4ByteTxRx()*. Adicionalmente, cada 0.1 segundos se realiza la lectura de la posición actual por medio de la función *read4ByteTxRx()*. La variable *posición actual* es necesaria en la ejecución de otros hilos que se encargan de graficar el radar y guardar las imágenes resultantes al hacer la detección. En caso de oprimir la tecla “ESC”, se deshabilita el torque del motor y se cierra el puerto. El algoritmo que lleva a cabo esta tarea se muestra en el anexo 8.2.2.

5.2.2. Revisión de modelos de redes neuronales

Teniendo en cuenta los trabajos revisados en la sección de estado del arte, se realiza una tabla (tabla 4) que describe las características principales de los modelos de segmentación.

Modelo	Descripción arquitectura	Característica relevante	Función de perdida	Dataset
U-net [65]	Arquitectura en forma de U. Con ruta de codificación y decodificación.	Tiene conexiones de salto para trasmitir información de alta resolución a las capas de decodificación.	Utiliza función de perdida suave de Dice la cual mide la similitud de la máscara de segmentación predicha y la de referencia.	Es eficaz para conjuntos de datos limitados debido a su capacidad para aprender características relevantes.
FCN [66] [37]	Compuesta por capas convolucionales por lo cual puede aceptar imágenes de cualquier tamaño y producir mapas de características de salida.	Puede hacer conexiones de salto directas entre las capas de codificación y las capas en la ruta de decodificación para preservar detalles finos.	Utiliza funciones de pérdida adecuada para la segmentación como la pérdida de entropía cruzada o la pérdida de Jaccard.	No necesita dataset grande. FCN se enfoca en segmentación semántica, obtiene buenos resultados con datasets limitados.
Mask-RCNN [29]	Realiza detección y segmentación semántica. Combina una red neuronal convolucional con una red de detección de objetos.	No incorpora conexiones de salto, pero, utiliza fusiones de características permitiendo capturar más información de diferentes niveles de resolución.	Utiliza una combinación de funciones de perdida, las cuales incluyen perdida de clasificación, perdida de regresión y perdida de segmentación.	Necesita un dataset de diferentes tamaños con anotaciones de segmentación y anotaciones de cuadros delimitadores.
DeepLab [67]	DeepLab aprovecha las convoluciones dilatadas y el componente ASPP (*Atrous Spatial Pyramid Pooling*) para capturar características contextuales a diferentes escalas.	No tienen conexión de salto, sin embargo, utiliza *atrous convolutions* en el decodificador para mejorar la precisión en la segmentación semántica	Es una combinación ponderada de la perdida de clasificación (*Cross-Entropy Loss*) y la perdida de regulación (*Regularization Losss*).	El dataset debe incluir imágenes anotadas con máscaras de segmentación detalladas y el dataset debe ser diverso.

TABLA 4: Características modelos de segmentación semántica

Para realizar la elección del modelo a utilizar en este proyecto específico se debe analizar el rendimiento de las diferentes arquitecturas de redes neuronales sobre unas métricas de interés. En la tabla 5 se resume las métricas para 5 modelos de redes neuronales.

5.2.3. Detección de objetos

En esta sección, se describe el proceso de selección, implementación y funcionamiento de las redes neuronales utilizadas tanto para la detección de objetos como para la segmentación semántica.

5.2.3.1. Métricas de evaluación

- IoU: Se utiliza el parámetro IoU (*Intersection over Union*), también conocido como *Jaccard Index*, para evaluar la calidad de la segmentación semántica. Esta métrica se basa en la comparación de la superposición entre la máscara de segmentación predicha por el modelo y la máscara de referencia o *ground truth*.

  La ecuación para calcular la métrica IoU en el proceso de segmentación semántica es la siguiente:

$$IoU = \frac{\text{Área de intersección}}{\text{Área de unión}} \tag{5.1}$$

- Precisión: La precisión es una métrica que evalúa qué tan exacto es un modelo en la tarea de segmentación semántica. Se calcula dividiendo el número de píxeles correctamente clasificados como positivos (verdaderos positivos) entre la suma de los verdaderos positivos y los falsos positivos. En otras palabras, la precisión muestra la proporción de predicciones positivas que son correctas. Esta métrica permite entender cuánta confianza se puede tener en las clasificaciones positivas realizadas por el modelo.

  La ecuación para calcular la precisión en la segmentación semántica es la siguiente:

$$\text{Precisión} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos Positivos}} \tag{5.2}$$

  En esta fórmula, los términos se definen de la siguiente manera:

  - Verdaderos Positivos (TP): Representa el número de píxeles correctamente clasificados como pertenecientes a la clase objetivo. Estos son los píxeles que se clasifican como positivos y realmente lo son.

Modelo	Detección o segmentación	Velocidad	Precisión	Capacidad de cómputo
U-net [65]	No incluye etapa de detección.	Al tener conexiones de salto, el proceso de inferencia se ralentiza ligeramente respecto a FCN.	Presenta buena precisión, especialmente en segmentación de estructuras biomédicas	Requiere menor capacidad de cómputo, ya que tiene menos capas.
SSD [68]	Enfoque de detección de objetos en tiempo real	No es más veloz que YOLO, pero es rápido en comparación con enfoques antiguos.	Realiza detección en varias escalas, lo que mejora la precisión respecto a modelos de un solo enfoque como YOLO.	Es eficiente en hardware con recursos limitados, lo que lo hace adecuado para implementaciones en dispositivos como sistemas embebidos o móviles.
FCN [66]	No está diseñada específicamente para detección sino para segmentación semántica.	Es más rápida porque su arquitectura simple y algoritmos más eficientes	Puede tener una precisión menor debido a que se encuentra especializada en aplicaciones en tiempo real.	FCN, es una arquitectura más simple, por tanto, es menos intensiva en recursos porque solo utiliza convoluciones completas.
Mask-RCNN [61]	Tiene la capacidad para abordar la segmentación y la detección de objetos en una sola arquitectura.	Tiende a ser un poco más lenta, ya que es una arquitectura en dos etapas.	Genera máscaras de segmentación detalladas y precisas debido a que combina las dos etapas (detección y segmentación) y realiza fusión de características.	Requiere mayor capacidad de cómputo por la arquitectura compleja y las operaciones relacionadas con detección y generación de máscaras.
DeepLab [67]	DeepLab se centra en segmentación semántica.	Puede ser más eficiente y rápida en términos de tiempo de ejecución y requerimientos computacionales.	Logra una segmentación precisa, pero puede tener dificultades para capturar detalles finos y bordes muy definidos debido a las convoluciones dilatadas.	Puede ser computacionalmente intensivo durante el entrenamiento y la inferencia, y se benefician de hardware potente.
YOLO [54]	Diseñado para detección, pero puede proporcionar una delimitación aproximada alrededor de los objetos detectados	Conocido por su velocidad. Puede lograr altas tasas de velocidad de procesamiento debido a su enfoque de detección única en toda la imagen.	Su precisión en ligeramente menor que otras arquitecturas, sin embargo, depende de la configuración y calidad del dataset.	Capaz de funcionar en hardware con recursos limitados. Es menos exigente en términos de capacidad de cómputo en comparación con Mask R-CNN.

TABLA 5: Comparación de modelos de redes neuronales según características del proyecto

- Falsos Positivos (FP): Indica el número de píxeles incorrectamente clasificados como pertenecientes a la clase objetivo. Estos son los píxeles que se clasifican como positivos, pero en realidad pertenecen a otras clases.

- Tiempo de ejecución: Se propone una métrica para evaluar el tiempo requerido para realizar la segmentación de una imagen en diferentes resoluciones. El objetivo principal de esta métrica es analizar el impacto de la resolución en el rendimiento del algoritmo de segmentación, es decir, cómo la variación en la resolución afecta el tiempo necesario para llevar a cabo la tarea de segmentación.

5.2.3.2. Elección de red

La elección de una red neuronal se realiza teniendo en cuenta las características específicas que tiene el proyecto. En este caso, la elección de la red neuronal se ve influenciada por factores como los recursos computacionales, las limitaciones de tiempo y la especialización de la red en detección de objetos. En la tabla 5 se evidencian el comportamiento de modelos como U-net, SSD (*Single Shot Detector*), FCN, Mask-RCNN, DeepLab y YOLO en aspectos como velocidad, precisión y capacidad de cómputo necesaria. Aunque otros modelos pueden destacar en cuanto a precisión y eficiencia de hardware, ninguno se compara en rapidez al desempeño de la red neuronal YOLO. Además, es importante destacar que el modelo YOLO no necesita un hadware complejo, sino que puede funcionar con recursos limitados. Dado que el objetivo del proyecto es identificar objetos, se opta por trabajar con el modelo YOLO debido a su enfoque en detección y localización de objetos, así como su eficiencia en tiempo real. En la sección 3.4 del capítulo 3 se profundiza en las especificaciones del modelo YOLO y en la sección 3.5 se da un contexto sobre las características principales de Mask R-CNN.

A partir de la tabla 1, se evidencia que el modelo de YOLOv7 reduce la cantidad de parámetros en un 39 %, las operaciones de punto flotante en 49 % y mantiene el porcentaje de precisión con respecto al modelo YOLOv4. Además, al analizar los resultados de YOLOv5 y YOLOv7-tiny se evidencia que YOLOv7 es 127 FPS más rápido y 10.7 % más preciso. Tras comparar distintas variantes de YOLO de la tabla 1 del capítulo 3, se opta por seleccionar la versión YOLOv7-Tiny. Esta elección se basa en su rendimiento óptimo, alta precisión y especialización en bordes. Además, se selecciona por ser un modelo optimizado para GPU, lo que implica una ejecución más eficiente y una adaptación adecuada al hardware disponible para el proyecto.

Por otra parte, en la tabla 4 se evidencian las características de modelos de red neuronal usados específicamente para segmentación semántica. En este proyecto, se utiliza el modelo preentrenado Mask R-CNN para la segmentación semántica debido a que ofrece alta precisión en la delimitación de objetos en comparación con otros modelos de detección como lo son DeepLab o FCN. Adicionalmente, es importante resaltar otros modelos de red neuronal se centran en un objetivo específico, ya sea detección o segmentación, en el caso de Mask-RCNN tiene la capacidad de abordar tanto segmentación semántica como detección de objetos en una sola arquitectura (tabla 5. Además, se elige debido a versatilidad para identificar y delimitar múltiples clases de objetos, lo cual resulta especialmente beneficioso en el entorno de prueba del laboratorio de robótica, donde pueden presentarse objetos de diferentes categorías. Asimismo, se selecciona este modelo debido a su disponibilidad y facilidad de uso a través de la biblioteca torchvision de PyTorch, lo que permite una implementación rápida y sencilla en el proyecto.

5.2.3.3. Implementación del modelo YOLOv7 preentrenado

El entrenamiento de modelos requiere grandes conjuntos de datos; sin embargo, crear un conjunto de datos específico para el desarrollo de este proyecto puede resultar insuficiente para lograr resultados óptimos. Por tanto, en este proyecto en particular, se ha optado por utilizar el conjunto de datos COCO y el modelo preentrenado de YOLOv7. La elección de un modelo preentrenado se basa en el hecho de que este tipo de modelos ya han aprendido patrones generales a partir de grandes conjuntos de datos y tareas relacionadas. Al aplicar el modelo preentrenado en el dominio de la detección de objetos, se espera obtener una mejor capacidad de respuesta y resultados más precisos.

En la figura 7, se indica el diagrama de flujo correspondiente a la implementación de modelo de detección de objetos YOLOv7 junto con el movimiento del motor y la implementación gráfica.

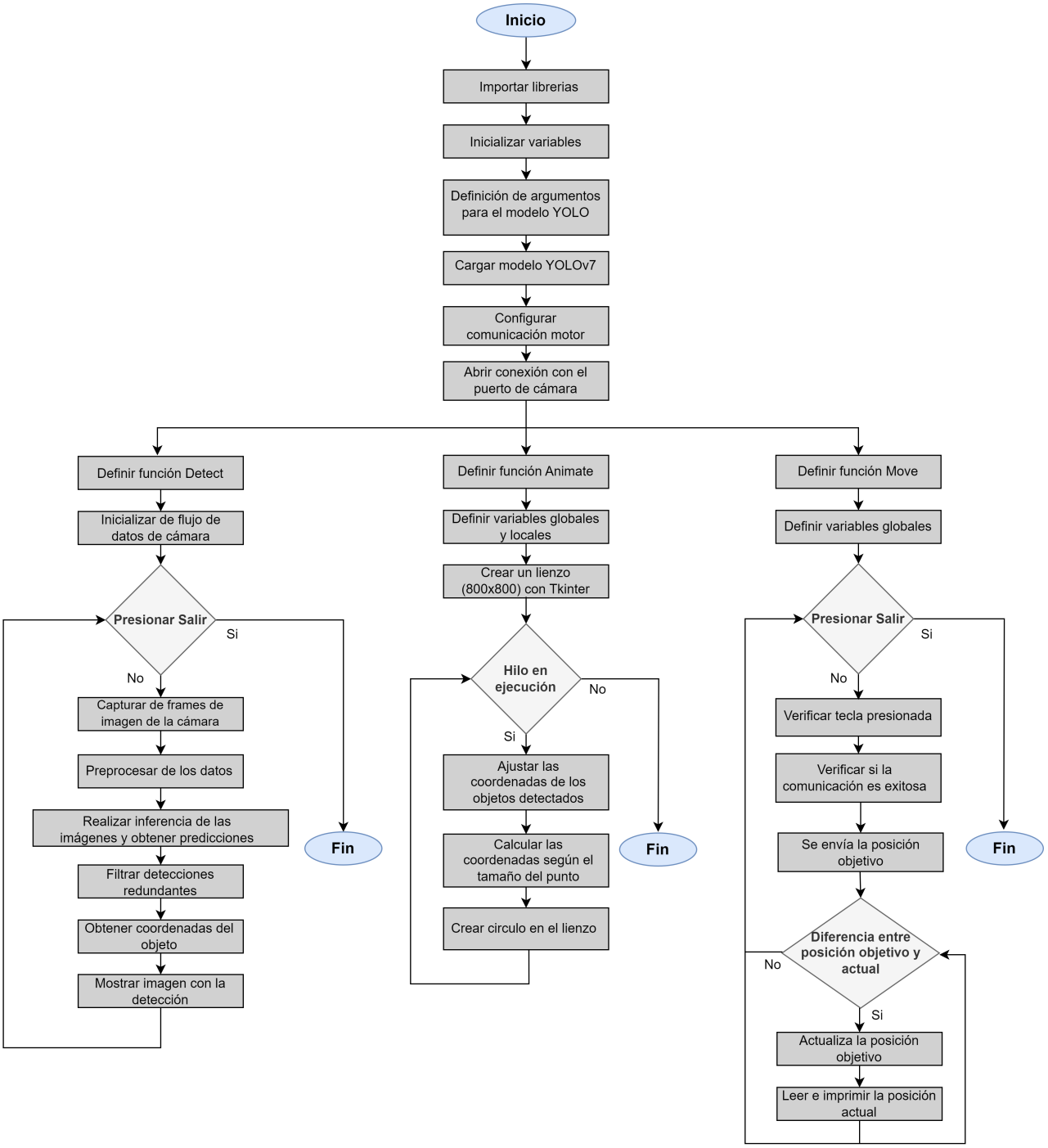

FIGURA 7: Diagrama de flujo código YOLO

Para implementar el modelo, en primer lugar se instalan paquetes necesarios y se descarga el modelo desde el repositorio de GitHub. En cuestión de algoritmo, se realiza la inicialización

del entorno de ejecución. A continuación, se carga el modelo de Yolo utilizando los pesos especificados y se verifica el tamaño de la imagen de tal manera que sea compatible con el modelo. Además, se obtiene los nombres de las clases que el modelo es capaz de detectar y asigna colores aleatorios a cada clase para su visualización.

Las clases con las que ha sido entrenado el modelo se muestran en la figura 4, las cuales hacen parte del conjunto de datos COCO. Se selecciona el conjunto de datos COCO, ya que proporciona imágenes con un contexto general que se adapta al laboratorio de robótica y no especifico como se pueden encontrar en otros conjuntos de datos.

- **Captura y preprocesamiento de imágenes:**
  Antes de realizar la inferencia del modelo se configura la captura de la imagen por medio de la cámara Intel RealSense. Las cámaras de profundidad Intel RealSense brindan datos dimensionales de alta precisión, esta característica permite medir objetos de cualquier tamaño o forma con facilidad, especialmente cuando se desea usar para el reconocimiento de objetos. Para utilizar la cámara por medio de Jetson Nano es necesario la biblioteca *pyrealsense2*. Una vez se instalan las librerías de la cámara, se configura las secuencias de la cámara tanto para color y profundidad utilizando los siguientes ajustes:

  - Resolución: 640x480
  - Formato píxeles para color: RGB8
  - Formato píxeles para profundidad: Z16
  - Velocidad de fotogramas 30 fps

  Posteriormente, se plantea un bucle *While* en el que se capturan continuamente los fotogramas de la cámara. Se realiza el preprocesamiento de las imágenes, el cual implica la alineación de los marcos de profundidad y color, seguido de la conversión de los marcos en una matriz. Además, se realiza la transformación de la imagen de profundidad en escala de grises a un mapa de color y se escalan los valores por un factor de 0.08 para que se ajusten al rango de valores de la imagen destino.

  El preprocesamiento se lleva a cabo con el objetivo de estandarizar y normalizar las imágenes, lo que permite tener una escala común para el procesamiento posterior. Además, la conversión de la imagen de profundidad a un mapa de color y el escalado de los valores tienen el propósito de mejorar la visualización y comprensión de la información en la imagen. Esto facilita la interpretación de los datos y proporciona una representación visual más intuitiva de la profundidad de la escena capturada por la cámara.

Por otro lado, YOLO utiliza una red neuronal convolucional con capas de submuestreo que reducen la resolución de la imagen en múltiplos de 2, por tanto, las imágenes deben ser redimensionadas al tamaño requerido por YOLO. También la imagen se normaliza dividiendo los valores de píxeles entre 255.

5.2.3.4. Detección

Una vez que se implementa el modelo y se ajustan los parámetros de la imagen de entrada, se procede a ingresar la imagen en el modelo de la red para llevar a cabo la detección.

- **Inferencia**
  En este punto, el modelo utiliza los parámetros aprendidos durante el entrenamiento para realizar cálculos y tomar decisiones basado en los datos de entrada. Al pasar la imagen a través de la red neuronal, las capas convolucionales de la red se encargan de extraer características relevantes de la imagen en diferentes niveles de abstracción. La salida de la red son mapas de características con información sobre la presencia y ubicación de objetos en la imagen. El resultado de la inferencia se guarda en una variable sobre la cual se aplica supresión de no máximos (NMS) para eliminar detecciones redundantes y obtener los objetos detectados finales.

- **Ajuste y visualización**
  Sobre las detecciones realizadas se realiza un escalamiento de las coordenadas del objeto detectado para ajustarlos al tamaño de la imagen original. Se verifican las etiquetas y se dibujan los cuadros delimitadores tanto en la imagen a color como en la imagen de profundidad utilizando la función *"plot_one_box"* como se muestran en la figura 8.

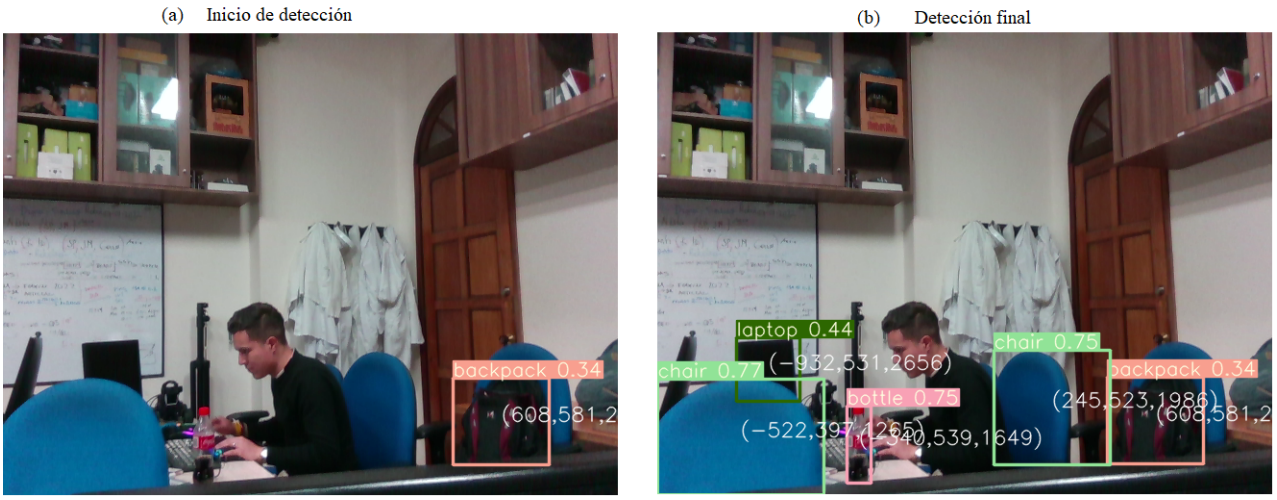

FIGURA 8: Visualización de cuadro de detección

5.2.3.5. Cálculo de distancias

Esta sección se encarga de calcular la posición tridimensional de un objeto detectado en la imagen. Para esto se calculan las coordenadas del punto central del cuadro delimitador del objeto detectado por medio de la fórmula:

$$x_{central} = (x_{izq} + x_{der})/2 \tag{5.3}$$

$$y_{central} = (y_{sup} + y_{inf})/2 \tag{5.4}$$

También se obtiene la distancia del objeto (*distancia*) utilizando la información de profundidad de la cámara. Para esto se utiliza la clase *"depth_frame"* al punto obtenido se le suma 4 en su coordenada *x* y 8 en su coordenada *y* para asegurar que la cámara de profundidad y la cámara RGB no se encuentren desalineadas. Para garantizar que la distancia calculada se encuentre en milímetros, se multiplica por 1000.

Para calcular las coordenadas tridimensionales del objeto se tiene en cuenta los valores calculados anteriormente junto con el punto principal del sensor de la cámara y la longitud focal tanto para el eje x como el eje y. Las formulas 5.3 y 5.4 indican el cálculo para las coordenadas de *x* y *y*.

$$X = distancia * (x_{central} + 4 - px) * fx - 35 \tag{5.5}$$

En la ecuación anterior, se indica el cálculo de la coordenada x del objeto detectado. La variable *distancia* representa la distancia del objeto, *px* es el punto principal del sensor de la cámara en el

eje x y *fx* corresponde la longitud focal en el eje x. Además, en la ecuación se resta 35 con el fin de ajustar la posición del objeto desde el centro de la cámara RGB hasta el centro de la cámara de profundidad. De manera similar se calcula la coordenada y:

$$Y = distancia * (y_{central} + 8 - py) * fy \tag{5.6}$$

Donde py corresponde al punto del sensor de la cámara y fy representa la longitud focal en el eje y. La coordenada tridimensional para Z es igual a la distancia del objeto.

5.2.4. Interfaz gráfica radar

En la rama central del diagrama de flujo de la figura 7 se indica el proceso para desarrollar la interfaz grafica. Inicialmente se establece el nombre de la ventana como "*mapping*" y se define el tamaño de la ventana con 800x850 píxeles. Se crean dos botones en la interfaz: "START" y "STOP" los cuales tienen asociada una función de comando asociada. El botón "START" inicia los tres hilos de ejecución simultáneos. Por su parte, el botón "STOP" se encarga de finalizar todo el programa cerrando la ventana de *mapping* y detiene el bucle principal. La visualización de las posiciones de los objetos se realiza en la función *animate*. Se utiliza la librería Tkinter y el elemento canvas para importar la imagen base del radar en la ventana de 800x800 píxeles. La interfaz gráfica se muestra en la figura 9.

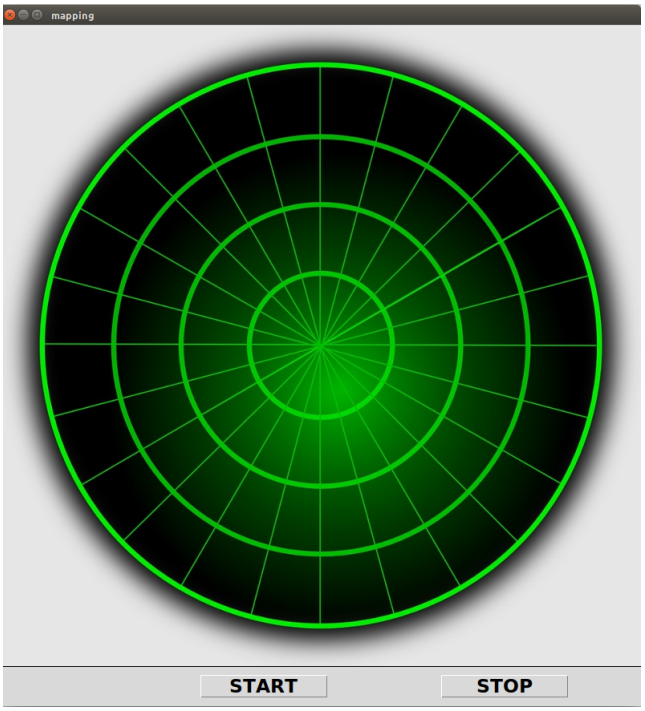

FIGURA 9: Interfaz gráfica base

La gráfica de la posición del objeto detectado se genera en función de las coordenadas y del estado del servomotor Dynamixel ax-12a. La posición de los puntos en la gráfica depende del valor de la variable *dxl_present_pos*, que representa la posición actual del motor. Los datos de las coordenadas de los objetos detectados se almacenan en la variable *object_coordinates*. Estos datos son sometidos a transformaciones y cálculos adicionales para determinar la posición exacta en la que se debe graficar el punto dentro del radar (canvas). Para esto, se convierte los valores de las coordenadas cartesianas en coordenadas polares por medio de las ecuaciones 5.5 y 5.6.

$$magnitud = \sqrt{x^2 + y^2} \tag{5.7}$$

$$fase = arctan(\frac{y}{x}) \tag{5.8}$$

El ángulo del punto detectado se escala entre 0° y 45°, para posteriormente ubicar las coordenadas en el centro del radar en canvas. Estos cálculos consideran las dimensiones y los desplazamientos necesarios para posicionar correctamente el punto en la gráfica. Finalmente, se crea un círculo con canvas utilizando las coordenadas calculadas.

5.2.5. Panorámica

En el código principal descrito en la sección 5.2.2, se capturan imágenes cada 30° por medio de la camara intel realsense. Estas imágenes solo proporcionan una vista por secciones del espacio en el cual se ejecuta el sistema, por tanto, con el fin de adquirir en una sola imagen una vista general del espacio, se unen las imágenes en una panorámica. Esto se realiza por medio de la función *stitcher* de OpenCv.

5.2.6. Segmentación semántica utilizando Mask R-CNN

La segmentación semántica realizada en el proyecto se basa en el uso del modelo Mask R-CNN con la arquitectura ResNet-50 preentrenada. Para realizar la implementación del modelo es necesario importar las librerías como opencv,*n*umpy y especialmente bibliotecas como *torch* para la infraestructura de la red neuronal y *functional torchvision* para transformación de imágenes. Luego, es necesario definir la ruta de las imágenes a segmentar, las cuales en este caso corresponden a las imágenes que fueron resultado de la detección en la sección 5.2.2.

- Cargar modelo: Para cargar una red preentrenada se crea una instancia del modelo que se desea utilizar. En este proyecto se crea una instancia del modelo Mask R-CNN que utiliza la arquitectura ResNet-50 con *Feature Pyramid Network*. El uso de esta red permite capturar características en diferentes escalas para mejorar la detección y segmentación de objetos en imágenes. Al crear la instancia se cargan los pesos preentrenados que permiten utilizar el modelo para la detección y segmentación específicamente. Adicionalmente, es necesario establecer el modelo en modo evaluación, esto con el fin de desactivar la etapa de entrenamiento y activar la etapa de inferencia.
- Preprocesamiento: La imagen de entrada se somete a un preprocesamiento para convertirla en un formato adecuado para la red neuronal. Esto incluye la conversión al formato RGB estándar y la transformación a una matriz multidimensional.
- Inferencia: La inferencia se realiza para una imagen a la vez y esta se pasa al modelo como una lista. Por su parte, el modelo devuelve predicciones utilizando técnicas de propuesta de regiones (*Region Proposal Network*), para identificar posibles regiones de interés en la imagen.
- Generación de máscaras de segmentación: En este punto, se extraen máscaras de segmentación y las etiquetas correspondientes para las regiones que se clasifican como objetos de

interés. Las máscaras de segmentación indican la ubicación precisa de cada objeto dentro de la región propuesta en la inferencia. Esto se realiza utilizando una rama de segmentación semántica que utiliza las características extraídas de la región propuesta para generar máscaras de píxeles. También se crea una máscara binaria para separar el fondo de las máscaras de los objetos y se segmenta los objetos de interés con un mismo color.

- Postprocesamiento de predicciones: Se combina la imagen original y la imagen resultante utilizando una operación de fusión ponderada, controlando la transparencia de los objetos resaltados mediante un parámetro llamado *alpha*. Además, se genera un nombre de archivo único para la imagen segmentada y se guarda la imagen. En la figura 11, se observa una imagen sin segmentar y una segmentada.

FIGURA 10: Imagen sin segmentar vs. imagen segmentada

A continuación, se presenta el diagrama de flujo en la figura 9, el cual muestra el desarrollo del algoritmo para realizar la segmentación semántica utilizando Mask R-CNN.

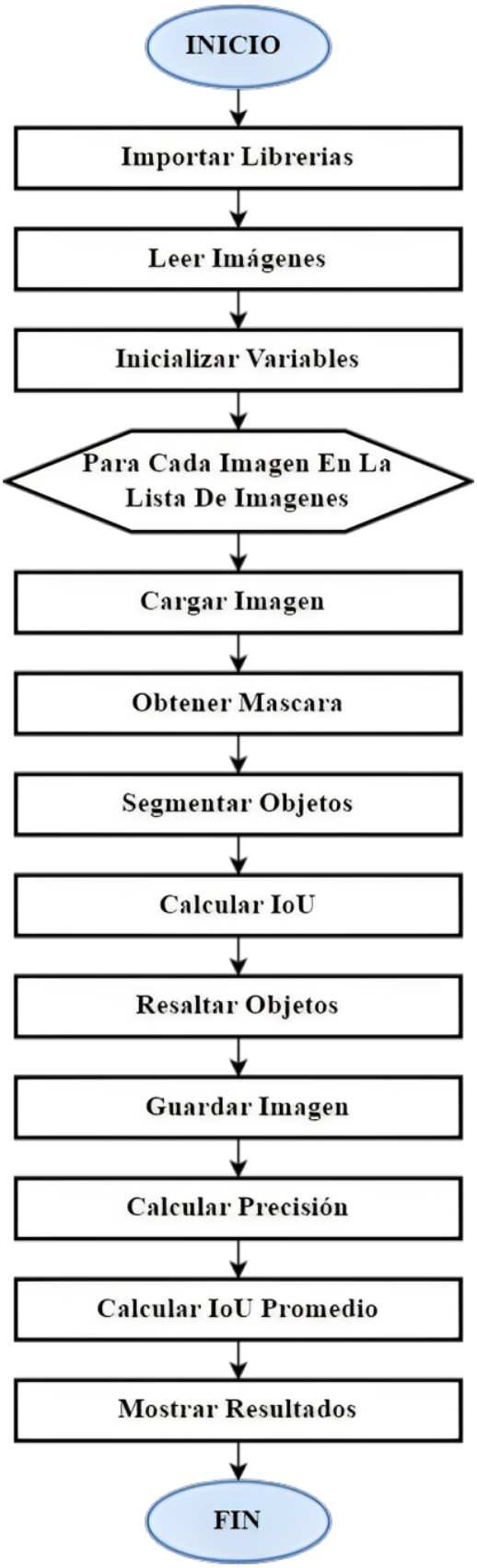

FIGURA 11: Diagrama de flujo del algoritmo Mask R-CNN

Después de realizar la segmentación semántica utilizando el modelo Mask R-CNN, es importante evaluar la calidad del resultado mediante métricas de evaluación. En la sección 5.2.2.1 del proyecto se encuentran las métricas utilizadas para evaluar la segmentación semántica.

Capítulo 6

Resultados y Discusión

6.1. Entrenamiento de red neuronal y utilización de redes preentrenadas

La visión artificial ha avanzado notablemente en áreas como la detección y la segmentación semántica. Estas tareas desafiantes que involucran la comprensión y el análisis de imágenes o videos han mostrado resultados impresionantes debido al poder de las redes neuronales profundas. Sin embargo, al considerar estos logros, surge una pregunta clave: ¿es más apropiado usar redes neuronales preentrenadas o entrenar redes neuronales para manejar estas tareas de manera más eficiente?

La transferencia de aprendizaje es un aspecto clave a considerar al utilizar un modelo preentrenado. Al entrenar un modelo desde cero para la segmentación semántica, se requeriría una gran cantidad de datos etiquetados y tiempo computacional significativo. En cambio, utilizar un modelo preentrenado permite aprovechar el conocimiento adquirido en tareas previas. Las características aprendidas por el modelo preentrenado, como la detección de bordes, texturas y patrones visuales, pueden ser útiles para la segmentación semántica, ya que también están presentes en esta tarea. Esto proporciona una ventaja inicial al modelo preentrenado, ya que no tiene que comenzar el aprendizaje desde cero.

Además, el modelo preentrenado ha aprendido a generalizar características relevantes en imágenes diversas. Esto significa que puede capturar patrones comunes en diferentes conjuntos de datos, lo que resulta en una mejor capacidad de generalización. Al aplicar el modelo preentrenado a la segmentación semántica en un nuevo conjunto de datos, se espera que pueda

reconocer y segmentar clases de objetos, incluso si no ha sido entrenado específicamente en ese dominio.

Sin embargo, es importante tener en cuenta las limitaciones de utilizar un modelo preentrenado. Estos modelos pueden estar sesgados hacia las características y clases presentes en el conjunto de datos utilizado durante el entrenamiento inicial. Si el dominio de las imágenes de la tarea de segmentación difiere significativamente del dominio en el que se entrenó el modelo preentrenado, su rendimiento puede verse afectado negativamente. En tales casos, puede ser necesario adaptar o ajustar el modelo preentrenado utilizando datos del dominio objetivo para mejorar su rendimiento en la segmentación semántica específica.

6.2. Pruebas del sistema

A partir de la implementación de las redes neuronales YOLOv7 y Mask R-CNN, se realizan pruebas del funcionamiento en diferentes entornos para evaluar la detección, la segmentación y el cálculo de la distancia en escenas que puedan presentar distintos objetos.

6.2.1. Pruebas de detección de objetos en diferentes escenas

Se realizan pruebas para verificar si la red neuronal realiza la detección de los objetos al interior de un espacio cerrado. La prueba se lleva a cabo en el laboratorio de robótica y las imágenes capturadas se indican en la figura 12. En este caso, se detectan personas, botellas, televisores, sillas, computadores portátiles y mesas.

FIGURA 12: Captura de imágenes respecto a posición del motor

A partir de las imágenes capturadas respecto a la posición del motor, se generó una imagen panorámica, la cual se muestra en la figura 13.

FIGURA 13: Imagen panorámica resultante

6.2.2. Pruebas del proceso de segmentación semántica

Para obtener los resultados se realizaron diferentes pruebas en espacios cerrados de la Universidad Santo Tomás, Sede principal, a continuación se presentan las pruebas y los resultados obtenidos.

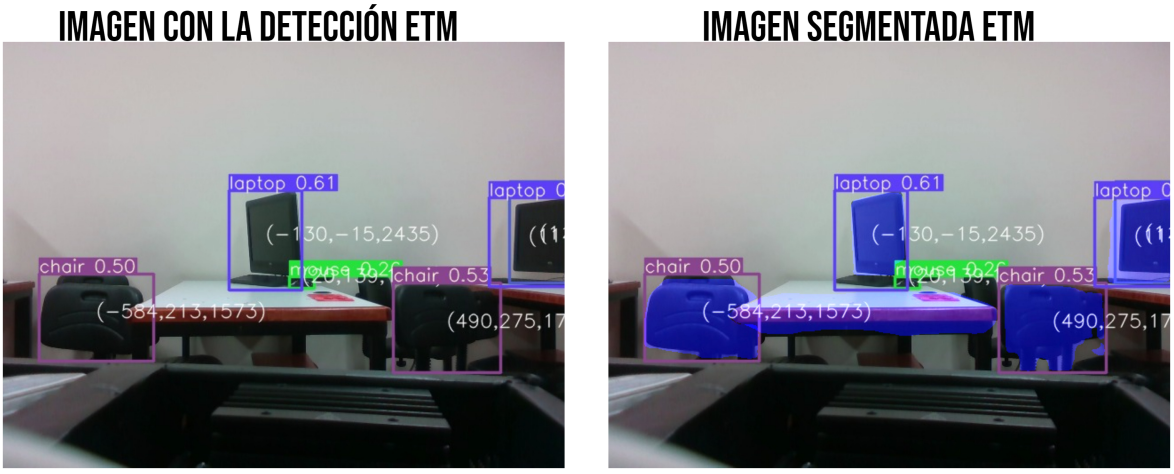

FIGURA 14: Imagen segmentada dentro de un laboratorio de electronica

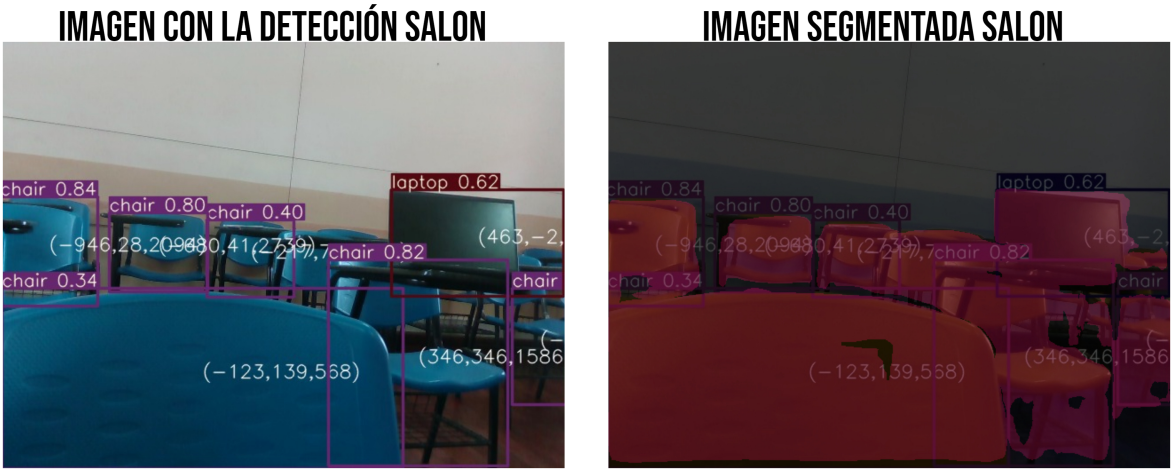

FIGURA 15: Imagen segmentada dentro de un salón de clase

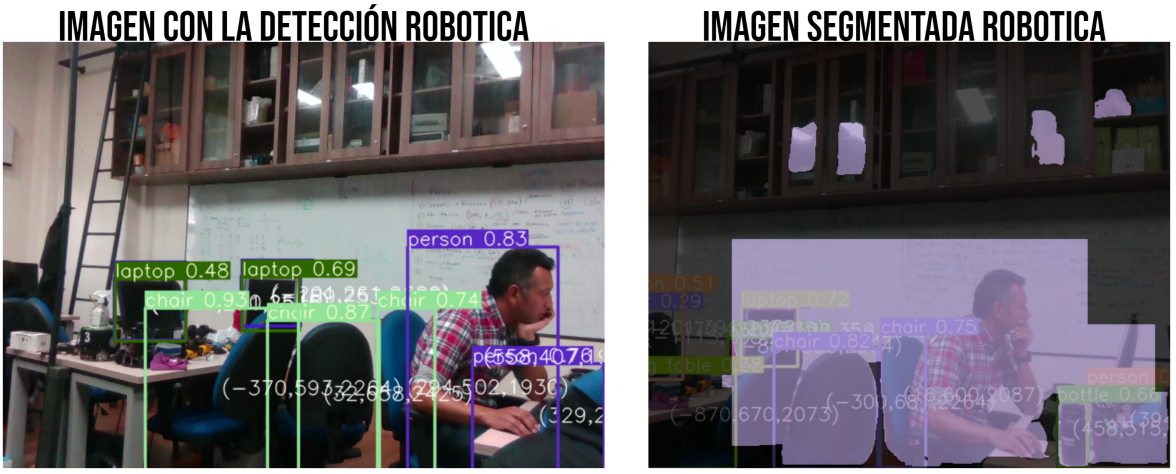

FIGURA 16: Imagen segmentada dentro del laboratorio de robotica

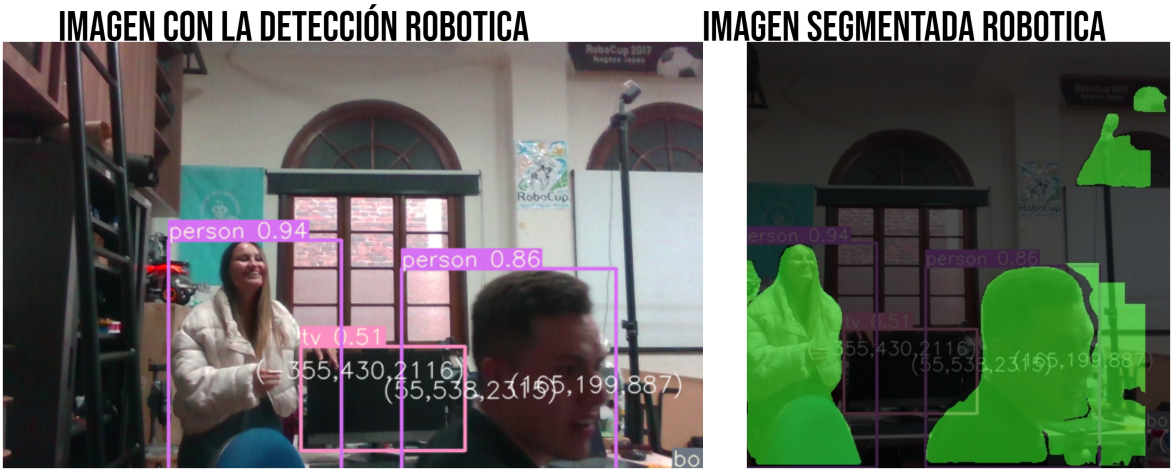

FIGURA 17: Imagen segmentada dentro del laboratorio de robótica

6.2.3. Cálculo de distancia

En la subsección 5.2.2.4 se indica la obtención de los valores de la posición de un objeto. Esto se realiza por medio de la imagen de profundidad de la cámara. En la figura 17 se muestra la imagen de profundidad que genera la cámara a diferentes distancias junto con la imagen de detección. Se evidencia que cuando la imagen de profundidad es de color azul oscuro, representa menor distancia entre el objeto y la cámara, en este caso, 0.625 metros (figura 17.a). Por otra parte, cuando la distancia entre la cámara y el objeto es igual a 1.38 metros, se puede considerar como una distancia media y la imagen de profundidad (figura 17.e) resalta al objeto

de color turquesa. Finalmente, se evidencian los resultados cuando la imagen de profundidad indica color naranja/rojo, lo cual implica que el objeto se encuentra a mayor distancia. La distancia entre el objeto y la cámara en este caso es de 2 metros, sin embargo, a partir de pruebas realizadas se evidenció que el sistema es capaz de detectar el valor de la distancia en valores hasta de 4 metros.

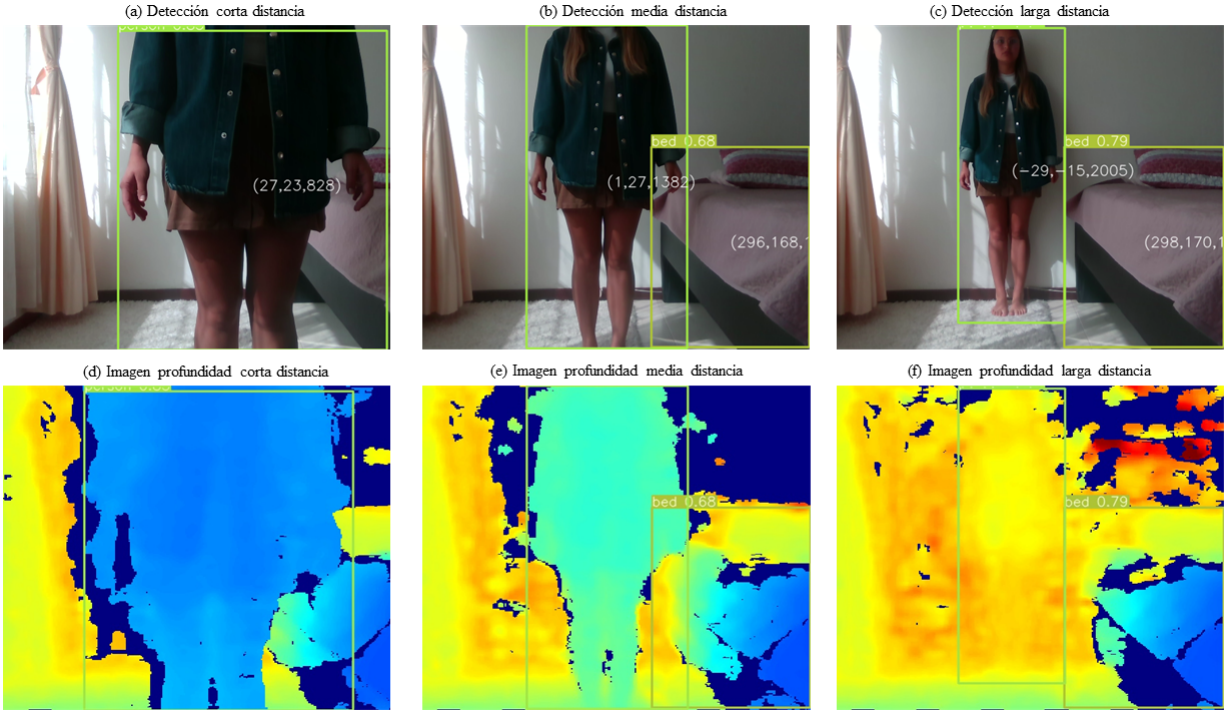

FIGURA 18: Variación de la distancia

6.3. Evaluación de los modelos

La evaluación de los dos modelos de redes neuronales se realiza teniendo en cuenta las métricas de evaluación definidas en la sección 5.2.2 a partir de la revisión bibliográfica.

6.3.1. Evaluación de la red YOLO

Durante las pruebas realizadas, se observa que la red exhibe cierto sesgo hacia las categorías de clase más prominentes del conjunto de datos COCO. La red es capaz de detectar objetos como personas, sillas, maletas, celulares, entre otros, incluso en entornos complicados. Sin embargo, en ocasiones llega a detectar objetos con etiquetas que no corresponden. Esta situación se debe

a que dentro del conjunto de datos, algunos objetos no se encuentran adecuadamente representados, mientras que otros objetos cuentan con cientos de anotaciones, como se evidencia en la figura 4. En el caso de los objetos poco representados como tostadoras y secadores de cabello, no se agregaron datos adicionales, ya que esas categorías no se consideran relevantes al no encontrarse en el contexto específico del proyecto.

Por otro lado, se evalúa la calidad de las predicciones realizadas usando el conjunto de prueba que forma parte de COCO. El test se lleva a cabo con umbral de confianza de 0.001, el umbral de superposición (IoU) de 0.65 y el tamaño del lote para la inferencia de 32 imágenes. Al observar

Modelo	AP	AP(50)	AP(75)
YOLOv5	28.0 %	45.7 %	27.0 %
YOLOv7-tiny	37.4 %	55.2 %	40.3 %

TABLA 6: Resultados comparativos de precisión entre YOLOv5 y YOLOv7

la proporción de predicciones correctas realizadas por el modelo, se observa que de todos los objetos presentes en las imágenes de prueba, el 37.4 % de ellos ha sido detectado y clasificado correctamente.

6.3.2. Evaluación Mask R-CNN

- **IoU**: En el contexto de este proyecto, la evaluación del rendimiento del valor promedio de IoU toma en consideración la complejidad del entorno en el que se lleva a cabo la segmentación, así como la naturaleza y características de los objetos bajo análisis. El código implementado procesa todas las imágenes almacenadas en una carpeta y lleva a cabo la segmentación de los objetos presentes en cada una de ellas. Sin embargo, debido a la variabilidad en la cantidad de objetos, su disposición en la escena y las condiciones de iluminación, es posible que se presenten desafíos adicionales en la correcta clasificación de algunos objetos.

  Cuando los objetos poseen características que dificultan su clasificación, como variaciones en la iluminación, tamaño, forma o la presencia de objetos superpuestos y parcialmente ocultos, el modelo de segmentación puede enfrentar desafíos para identificarlos correctamente, lo que puede afectar la precisión del IoU.

  En el caso del algoritmo de segmentación, el valor de IoU promedio obtenido es del 7.85 %. Dentro de este contexto, un IoU promedio tan bajo puede indicar que la segmentación realizada por el modelo puede no ser lo suficientemente precisa para capturar con precisión los contornos y características de los objetos en algunas imágenes.

- **Precisión:** El código de segmentación semántica utilizado ha obtenido una precisión del 72.97 %. Esto indica que una proporción significativa de los píxeles clasificados como objetos de interés fueron correctamente identificados por el modelo. Además de la precisión obtenida, existen varios aspectos que se pueden considerar para mejorar la precisión de la segmentación semántica:

  - Mejora de los datos de entrenamiento: Los resultados de la segmentación semántica están influenciados por la cantidad de los datos de entrenamiento. Asegurarse de tener un conjunto de datos diverso y representativo, con anotaciones precisas, puede ayudar a mejorar la precisión del modelo. Además, la recopilación de más datos y la incorporación de datos adicionales pueden ser beneficiosas para obtener un mejor rendimiento.
  - Consideración de arquitecturas más avanzadas: El modelo utilizado en el código es el Mask R-CNN con la arquitectura ResNet-50. Sin embargo, existen arquitecturas más avanzadas y sofisticadas que podrían mejorar aún más la precisión, como las redes neuronales convolucionales profundas (CNN).
  - Postprocesamiento de resultados: Aplicar técnicas de postprocesamiento a los resultados de la segmentación, como filtrado de ruido, suavizado o eliminación de falsos positivos, puede ayudar a mejorar la precisión final.

- **Tiempo de ejecución:** Para llevar a cabo la evaluación, se seleccionó una imagen de referencia y se realizó la segmentación en tres resoluciones diferentes: (320, 240), (640, 480) y (1280, 720). Los resultados obtenidos muestran los tiempos de ejecución correspondientes a cada una de estas resoluciones. A continuación, se presenta en la figura 19 la imagen de referencia

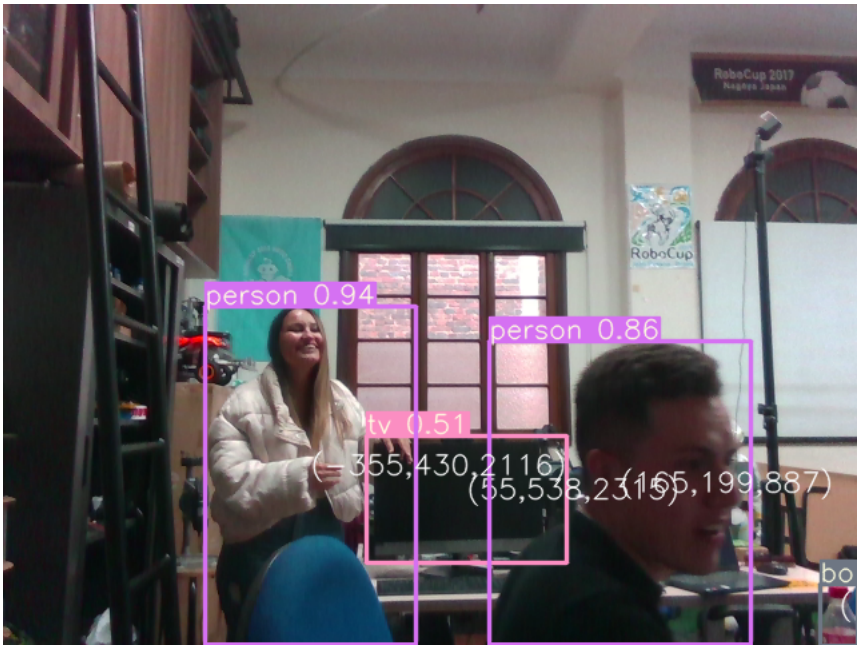

FIGURA 19: Figura de referencia

En el caso de la resolución (320, 240), se observó que el tiempo de ejecución fue de 8.166 segundos. Posteriormente, al evaluar la misma imagen en una resolución de (640, 480), se obtuvo un tiempo de ejecución de 9.190 segundos. Por último, al considerar una resolución de (1280, 720), se registró un tiempo de ejecución de 10.001 segundos, A continuación, se presenta en la figura 20 y 21 los resultados con la imagen de referencia.

FIGURA 20: Imagen segmentada con las diferentes resoluciones (320, 240) y (640, 480)

FIGURA 21: Imagen segmentada con la última resolución (1280, 720)

Estos resultados revelan una tendencia en la cual el tiempo de ejecución tiende a aumentar a medida que se incrementa la resolución de la imagen. Esto indica que, en general, segmentar imágenes de mayor resolución requiere más tiempo de procesamiento por parte del algoritmo.

Capítulo 7

Conclusiones y Trabajos futuros

- A partir de la revisión bibliográfica realizada, se ha encontrado evidencia de que las diversas arquitecturas de redes neuronales convolucionales utilizadas en la segmentación semántica son evaluadas con base a parámetros como precisión, el Índice de Jaccard (IoU) y tiempo (tabla 4 y tabla 1). Sin embargo, al considerar las limitaciones de la tarjeta Jetson Nano, la velocidad de la red y su capacidad de memoria RAM limitada, se ha llegado a la conclusión de que YOLO emerge como una opción altamente recomendada para proyectos que requieren una detección rápida, al tiempo que se destaca por su eficiencia computacional.

- Tras realizar un análisis y considerar las métricas y la capacidad de memoria de la tarjeta Jetson, se concluye que la integración de los algoritmos YOLO y Mask R-CNN proporciona una solución integral y altamente efectiva para el reconocimiento y manipulación de objetos en el entorno del laboratorio de robótica. Lo anterior se muestra en la figura 17 donde se muestra que la detección (imagen izquierda) funciona como un complemento de la segmentación (imagen derecha). El uso de YOLO en la etapa de detección de objetos permite una rápida identificación en tiempo real, mientras que el algoritmo de Mask R-CNN para la segmentación semántica garantiza una representación visual precisa de los objetos de interés.

- Al evaluar la segmentación utilizando métricas como la Precisión y el Índice de Jaccard (IoU), se obtuvieron los resultados de la sección 6.3.2 donde se evidencia que: la precisión del 72.97 % indica que una proporción significativa de los píxeles clasificados como objetos de interés fue identificada correctamente por el modelo. Sin embargo, es importante tener en cuenta que existen ciertos factores que pueden influir en la precisión de la segmentación. Además, debido a la posible similitud entre algunos objetos, puede haber

superposición parcial entre las máscaras de objeto y la máscara de referencia, lo que se refleja en el valor de IoU del 7.85 %. Estos resultados sugieren que, si bien el modelo logra identificar la mayoría de los objetos de interés, puede haber variaciones en la precisión en términos de la coincidencia exacta de los objetos. Es importante considerar el impacto del brillo, la distancia y las características que pueden llevar a una segmentación menos precisa en ciertos casos.

- El algoritmo desarrollado no solo es capaz de realizar detección y segmentación de objetos, sino que también se ha logrado integrar de manera práctica en la ejecución de un radar mediante el movimiento de un motor y la representación gráfica de las posiciones de los objetos. Además, este proyecto tiene un potencial futuro para ser implementado en otras aplicaciones en la cuales se pueda aprovechar la capacidad de identificar la posición de un objeto dentro de un espacio.

El modelo propuesto en este trabajo de grado se integrará en proyectos de investigación dedicados a la búsqueda y rescate de víctimas en zonas de desastre como se muestra en [69],[70],[71] y [72]. También se utilizará en el sistema de navegación de plataformas robóticas desarrolladas por el grupo de investigación con aplicaciones de asistencia doméstica [20], cuidado y desinfección de zonas comunes [73], o transporte de carga en sistemas multirobot [74] y [75].

Capítulo 8

Anexos

8.1. Código Segmentación Semántica Mask R-CNN

1. Importar Librerías.

```
import os
import cv2
import numpy as np
import torch
from torchvision.models.detection import maskrcnn_resnet50_fpn
from torchvision.transforms import functional as F
import random
```

2. Se llama al modelo Mask R-CNN.

```
model = maskrcnn_resnet50_fpn(pretrained=True)
model.eval()
```

3. Se especifica la ruta de la carpeta que contiene las imágenes a segmentar en la variable *"path"*.

```
path = path ="/home/jetson/yolov7"
```

4. Se obtiene la lista de nombres de archivo de las imágenes en la carpeta especificada. Solo se seleccionan los archivos con la extensión "png".

```
img_extension = ".png"

# Obtener la lista de nombres de archivo en la carpeta
files = os.listdir(path)
images = [os.path.join(path, file) for file in files if os.path.splitext(file)[1].lower() == img_extension]
```

5. Se crea una lista de etiquetas correspondientes a los objetos que se desean resaltar en la segmentación. Estas etiquetas se especifican en la variable `objetos_interes`.

```
objetos_interes = [1, 2, 3, 4, 7, 73]
```

6. Se inicializan las variables necesarias para el cálculo de la intersección sobre unión (IoU) y precisión.

```
ious = []
tp_sum = 0
fp_sum = 0
```

7. Dentro del bucle, se procesa cada imagen, realizando una serie de funciones para obtener los resultados deseados. Estas funciones incluyen cargar la imagen, obtener la máscara, segmentar los objetos, calcular métricas y guardar los resultados obtenidos.

   - Cargar imagen: Se lee la imagen utilizando *cv2.imread* y se almacena en la variable *image*.

```
image = cv2.imread(image_path)
```

   - Obtener la máscara: Primero, se realiza una conversión de la imagen al espacio de color RGB. A continuación, se lleva a cabo una transformación de la imagen en un formato adecuado utilizando las funciones proporcionadas por la biblioteca *torchvision.transforms.function* Esta transformación permite preparar la imagen para su posterior procesamiento. Posteriormente, el modelo se utiliza para identificar y generar las máscaras de los objetos detectados en la imagen.

```
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image_tensor = F.to_tensor(image_rgb)

with torch.no_grad():
    predictions = model([image_tensor])

masks = predictions[0]["masks"].detach().cpu().numpy()
labels = predictions[0]["labels"].detach().cpu().numpy()
```

   - Segmentar los objetos: Se segmentan los objetos de interés utilizando las máscaras obtenidas y se asigna un color aleatorio a cada objeto.

```
for mask, label in zip(masks, labels):
    if label in objetos_interes:
        mask = np.squeeze(mask, axis=0).astype(np.bool)
        segmented_image[mask] = object_color
```

- Calcular métricas: Se calcula la intersección sobre unión (IoU) y se actualizan las variables de conteo para el cálculo de la precisión.

```
object_color = [random.randint(0, 255) for _ in range(3)]
segmented_image = np.zeros_like(image)
for mask, label in zip(masks, labels):
    if label in objetos_interes:
        mask = np.squeeze(mask, axis=0).astype(np.bool)
        segmented_image[mask] = object_color

        intersection = np.logical_and(mask, np.squeeze(background_mask >= 0.5, axis=0))
        intersection_area = np.count_nonzero(intersection)

        union = np.logical_or(mask, np.squeeze(background_mask >= 0.5, axis=0))
        union_area = np.count_nonzero(union)

        iou = intersection_area / union_area
        ious.append(iou)


        tp_sum += np.count_nonzero(intersection)


        fp_sum += np.count_nonzero(mask) - intersection_area
```

- Guardar los resultados: Se combina la imagen original con la imagen segmentada y resaltada. La imagen resultante se guarda en una nueva ruta con un nombre único.

```
filename = os.path.splitext(os.path.basename(image_path))[0]
saved_path = os.path.join(path, filename + "_segmentacion.jpg")

cv2.imwrite(saved_path, result_image)
print("Imagen segmentada y resaltada guardada en:", saved_path)
```

Se repiten los pasos anteriores para cada imagen en la lista.

8. Se calcula la precisión dividiendo el número de verdaderos positivos entre la suma de verdaderos positivos y falsos positivos. El resultado se almacena en la variable precisión.

```
precision = tp_sum / (tp_sum + fp_sum)
```

9. Se calcula el promedio de las intersecciones sobre uniones (IoU) obtenidas en el proceso y se almacena en la variable.

```
mean_iou = np.mean(ious)
```

10. Se imprime en pantalla la precisión y el IoU promedio obtenidos.

```
print("Precisión: {:.2%}".format(precision))
print("IoU promedio: {:.2%}".format(mean_iou))
```

8.2. Código detección de objetos Yolo

En esta sección se encuentran los algoritmos correspondientes a la implementación del modelo YOLO, el movimiento del motor y la representación gráfica.

8.2.1. Implementación del modelo

1. Importar librerías para hacer la detección de objetos.

```
import argparse
import time
import pyrealsense2 as rs
import numpy as np
import cv2
import torch
import torch.backends.cudnn as cudnn
import tkinter as tk
import tkinter.font as font
import threading
import os
import pyrealsense2 as rs
import sys

from dynamixel_sdk import *
from numpy import random
from pathlib import Path
from models.experimental import attempt_load
from utils.general import check_img_size, check_requirements, check_imshow, non_max_suppression, apply_classifier, \
    scale_coords, xyxy2xywh, strip_optimizer, set_logging, increment_path
from utils.plots import plot_one_box
from utils.torch_utils import select_device, load_classifier, time_synchronized, TracedModel
from PIL import ImageTk, Image, ImageGrab
```

2. Procesar los argumentos de la línea de comandos

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default='yolov7.pt', help='model.pt path(s)')
    parser.add_argument('--source', type=str, default='inference/images', help='source')
    parser.add_argument('--img-size', type=int, default=640, help='inference size (pixels)')
    parser.add_argument('--conf-thres', type=float, default=0.25, help='object confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='IOU threshold for NMS')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='display results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
    parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --class 0, or --class 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default='runs/detect', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--no-trace', action='store_true', help='don`t trace model')
    opt = parser.parse_args()
    print(opt)
```

3. Se carga el modelo

```
if trace:
    model = TracedModel(model, device, opt.img_size)
if half:
    model.half()  # to FP16
```

4. Definir la función *"detect"* que realizará la detección de objetos.

```
def detect(save_img=True):
```

5. Asignar los valores de los argumentos de línea de comandos a variables locales.

```
source, weights, view_img, save_txt, imgsz, trace = opt.source, opt.weights, opt.view_img,
opt.save_txt, opt.img_size, not opt.no_trace
save_img = not opt.nosave and not source.endswith('.txt')  # save inference images
webcam = source.isnumeric() or source.endswith('.txt') or source.lower().startswith(
    ('rtsp://', 'rtmp://', 'http://', 'https://'))
```

6. crear directorios para guardar los resultados de la detección.

```
save_dir = Path(increment_path(Path(opt.project) / opt.name, exist_ok=opt.exist_ok))
(save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True)
```

7. Inicializar la configuración.

```
set_logging()
device = select_device(opt.device)
half = device.type != 'cpu'
```

8. Cargar el modelo de detección.

```
model = attempt_load(weights, map_location=device)  # load FP32 model
stride = int(model.stride.max())  # model stride
imgsz = check_img_size(imgsz, s=stride)  # check img_size
```

9. Configurar los datos*"dataset"* según la fuente de entrada.

```
if webcam:
    view_img = check_imshow()
    cudnn.benchmark = True  # set True to speed up constant image size inference
    dataset = LoadStreams(source, img_size=imgsz, stride=stride)
else:
    dataset = LoadImages(source, img_size=imgsz, stride=stride)
```

10. Obtenerlos nombres de las clases y generar colores aleatorios para cada clase.

```
names = model.module.names if hasattr(model, 'module') else model.names
colors = [[random.randint(0, 255) for _ in range(3)] for _ in names]
```

11. Realizar la inferencia en cada imagen.

```
for path, img, im0s, vid_cap in dataset:
    img = torch.from_numpy(img).to(device)
    img = img.half() if half else img.float()
    img /= 255.0  # 0 - 255 to 0.0 - 1.0
    if img.ndimension() == 3:
        img = img.unsqueeze(0)
```

12. Realizar la inferencia en el modelo.

```
with torch.no_grad():
    pred = model(img, augment=opt.augment)[0]
t2 = time_synchronized()
```

13. Aplicar la clasificación.

```
if classify:
    pred = apply_classifier(pred, modelc, img, im0s)
```

14. Procesar las detecciones y guardar los resultados.

```
for i, det in enumerate(pred):  # detections per image
    if webcam:  # batch_size >= 1
        p, s, im0, frame = path[i], '%g: ' % i, im0s[i].copy(), dataset.count
    else:
        p, s, im0, frame = path, '', im0s, getattr(dataset, 'frame', 0)

    p = Path(p)  # to Path
    save_path = str(save_dir / p.name)  # img.jpg
    txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image' else f'_{frame}')
    gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
    if len(det):
        # Rescale boxes from img_size to im0 size
        det[:, :4] = scale_coords(img.shape[2:], det[:, :4], im0.shape).round()

        # Print results
        for c in det[:, -1].unique():
            n = (det[:, -1] == c).sum()  # detections per class
            s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string
```

15. Mostrar y guardar los resultados

```
if view_img:
    cv2.imshow(str(p), im0)
    cv2.waitKey(1)  # 1 millisecond

# Save results (image with detections)
if save_img:
    if dataset.mode == 'image':
        cv2.imwrite(save_path, im0)
        print(f" The image with the result is saved in: {save_path}")
    else:  # 'video' or 'stream'
        if vid_path != save_path:  # new video
            vid_path = save_path
            if isinstance(vid_writer, cv2.VideoWriter):
                vid_writer.release()  # release previous video writer
            if vid_cap:  # video
                fps = vid_cap.get(cv2.CAP_PROP_FPS)
                w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
            else:  # stream
                fps, w, h = 30, im0.shape[1], im0.shape[0]
                save_path += '.mp4'
            vid_writer = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
        vid_writer.write(im0)
```

16. Imprimir la información sobre el tiempo de ejecución y los resultados

```
if save_txt or save_img:
    s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir / 'labels'}" if save_txt else ''
```

17. Incluir el bloque principal de ejecución

```
with torch.no_grad():
    if opt.update:  # update all models (to fix SourceChangeWarning)
        for opt.weights in ['yolov7.pt']:
            detect()
            strip_optimizer(opt.weights)
    else:
        detect()
```

8.2.2. Giro del motor

1. Se definen las direcciones de la tabla de control y otros parámetros relacionados con el control del motor.

```
ADDR_DXL_CW_ANGLE_LIMIT    = 6
ADDR_DXL_CCW_ANGLE_LIMIT   = 8
ADDR_DXL_DRIVE_MODE        = 10
ADDR_MX_TORQUE_ENABLE      = 24
ADDR_MX_GOAL_POSITION      = 30
ADDR_DXL_GOAL_SPEED        = 32
ADDR_MX_PRESENT_POSITION   = 36
ADDR_DXL_PRESENT_SPEED     = 38

# Protocol version
PROTOCOL_VERSION            = 1.0

# Default setting
DXL_ID                      = 1
BAUDRATE                    = 1000000
DEVICENAME                  = '/dev/ttyUSB0'


TORQUE_ENABLE               = 1
TORQUE_DISABLE              = 0
DXL_MINIMUM_POSITION_VALUE  = 0
DXL_MAXIMUM_POSITION_VALUE  = 1023
DXL_MOVING_STATUS_THRESHOLD = 5
```

2. Se inicializan los controladores y se abre el puerto de comunicación con el motor.

```
index = 0
dxl_goal_position = 0

portHandler = PortHandler(DEVICENAME)

packetHandler = PacketHandler(PROTOCOL_VERSION)


# Open port
if portHandler.openPort():
    print("Succeeded to open the port")
else:
    print("Failed to open the port")
    print("Press any key to terminate...")
    getch()
    quit()


# Set port baudrate
if portHandler.setBaudRate(BAUDRATE):
    print("Succeeded to change the baudrate")
else:
    print("Failed to change the baudrate")
    print("Press any key to terminate...")
    getch()
    quit()
```

3. Se habilita el torque en el motor Dynamixel.

```
# Enable Dynamixel Torque
dxl_comm_result, dxl_error = packetHandler.write1ByteTxRx(portHandler, DXL_ID, ADDR_MX_TORQUE_ENABLE, TORQUE_ENABLE)
if dxl_comm_result != COMM_SUCCESS:
    print("%s" % packetHandler.getTxRxResult(dxl_comm_result))
elif dxl_error != 0:
    print("%s" % packetHandler.getRxPacketError(dxl_error))
else:
    print("Dynamixel has been successfully connected")
```

4. Configuración inicial de la función

```
def move():
    global ADDR_DXL_CW_ANGLE_LIMIT, ADDR_DXL_CCW_ANGLE_LIMIT, ADDR_DXL_DRIVE_MODE, ADDR_MX_GOAL_POSITION, ADDR_DXL_GOAL_SPEED,
    ADDR_MX_PRESENT_POSITION, ADDR_DXL_PRESENT_SPEED, PROTOCOL_VERSION, DXL_MOVING_STATUS_THRESHOLD
    global BAUDRATE, DEVICENAME, TORQUE_DISABLE, TORQUE_ENABLE, DXL_MINIMUM_POSITION_VALUE, DXL_MAXIMUM_POSITION_VALUE,
    portHandler, packetHandler,  dxl_present_pos,  dxl_goal_position, flag
```

5. Bucle principal

```
while 1:
    print("Press any key to continue! (or press ESC to quit!)")
    if getch() == chr(0x1b):
        break
```

6. Escribir la posición objetivo

```
dxl_comm_result, dxl_error = packetHandler.write4ByteTxRx(portHandler, DXL_ID, ADDR_MX_GOAL_POSITION, dxl_goal_position)
if dxl_comm_result != COMM_SUCCESS:
    print("%s" % packetHandler.getTxRxResult(dxl_comm_result))
elif dxl_error != 0:
    print("%s" % packetHandler.getRxPacketError(dxl_error))
```

7. Leer la posición actual

```
while 1:
    # Read present position
    time.sleep(0.1)
    dxl_present_pos, dxl_comm_result, dxl_error = packetHandler.read4ByteTxRx(portHandler, DXL_ID, ADDR_MX_PRESENT_POSITION)
    if dxl_comm_result != COMM_SUCCESS:
        print("%s" % packetHandler.getTxRxResult(dxl_comm_result))
    elif dxl_error != 0:
        print("%s" % packetHandler.getRxPacketError(dxl_error))

    print("[ID:%03d] GoalPos:%03d  PresPos:%03d" % (DXL_ID, dxl_goal_position, dxl_present_pos))

    if not abs(dxl_goal_position - dxl_present_pos) > DXL_MOVING_STATUS_THRESHOLD:
        flag = 1
        break
```

8. Cambiar la posición objetivo

```
if dxl_goal_position <= 920:
    dxl_goal_position += 100
elif dxl_goal_position >= 1000:
    dxl_goal_position = 0
```

9. Desactivar el torque del motor y cerrar el puerto

```
dxl_comm_result, dxl_error = packetHandler.write1ByteTxRx(portHandler, DXL_ID, ADDR_MX_TORQUE_ENABLE, TORQUE_DISABLE)
if dxl_comm_result != COMM_SUCCESS:
    print("%s" % packetHandler.getTxRxResult(dxl_comm_result))
elif dxl_error != 0:
    print("%s" % packetHandler.getRxPacketError(dxl_error))

# Close port
portHandler.closePort()
```

8.2.3. Radar

1. Se definen las variables globales: *object_coordinates* , *delete_objects* , *img_tk* y *dxl_present_pos*.

```
def animate():
    global object_coordinates, delete_objects, img_tk, dxl_present_pos
```

2. Se establece *dxl_present_pos* en 0, inicializando así la posición actual del motor Dinamixel, además de definir *circle_radius* , y *scale* , que son variables para ajustar el tamaño de los círculos y la escala de animación.

```
dxl_present_pos = 0
circle_radius = 6
scale = 15 # mm/pixel
```

3. Se crea una imagen en el lienzo *canvas1* en la posición (400,400)utilizando la imagen *img_tk*.

```
canvas1.create_image(400,400, image=img_tk)
```

4. Bucle principal este se ejecuta mientras la variable *do_run* del hilo de ejecución actual ("t") sea verdadera.

```
while getattr(t, "do_run", True):
    canvas1.create_image(400,400, image=img_tk)
    for i in range(len(object_coordinates)):
        cx = object_coordinates[i][1]/4
        cy = object_coordinates[i][2]/scale
        print("X: %d,  Y: %d" % (cx,cy))
```

5. Se calcula la magnitud *"mag"* y el ángulo *angle* utilizando las coordenadas *cx* y *cy*. También se calcula el ángulo del motor Dinamixel *angle_mot* utilizando su posición actual.

```
mag = np.sqrt(cx**2 + cy**2)
angle = np.degrees(np.arctan(cy/cx))/4
angle_mot = (dxl_present_pos * 360)/1023
```

6. Se calculan las nuevas coordenadas *"newX y newY"* de los objetos ajustando el ángulo de acuerdo al ángulo del motor Dynamixel.

```
newX = mag*np.cos(np.radians(angle - angle_mot))+400
newY = mag*np.sin(np.radians(angle - angle_mot))+400
flag = 1
```

7. Se calculan las coordenadas del cuadro del círculo *"X1","Y1","X2","Y2"* utilizando las nuevas coordenadas y el radio del círculo.

```
x1 = newX - circle_radius
x2 = newX + circle_radius
y1 = newY - circle_radius
y2 = newY + circle_radius
circle1 = canvas1.create_oval(x1, y1, x2, y2, fill = "green", tag=str(i))
```

8. Se crea un círculo nuevo en el lienzo *canvas1* y se va actualizando.

```
canvas1.coords(circle1, x1, y1, x2, y2)
```

9. Si el ángulo del motor Dynamixel *angle_mot* supera un umbral ("365"), se elimina el círculo anterior creado, se borran los objetos almacenados en *object_coordinates* y *delete_objects* en falso.

```
if angle_mot >= 356:
    canvas1.delete(circle1)
    object_coordinates.clear()
    delete_objects=False
```

Bibliografía

[1] James et al. *Image Processing and Computer Vision*. 1999. DOI: 10.1007/0-387-24579-0_5. URL: https://doi.org/10.1007/0-387-24579-0_5.

[2] Pedro Meseguer Gonzalez y Ramon Lopez de Mantaras Badia. *Inteligencia artificial*. Editorial CSIC Consejo Superior de Investigaciones Cientificas, 2017, pág. 159. ISBN: 9788400102340. URL: https://elibro.net/es/lc/usta/titulos/42319.

[3] Xingchao Yan et al. «RAFNet: RGB-D attention feature fusion network for indoor semantic segmentation». En: *Displays* 70 (2021), pág. 102082. ISSN: 0141-9382. DOI: https://doi.org/10.1016/j.displa.2021.102082. URL: https://www.sciencedirect.com/science/article/pii/S0141938221000883.

[4] Jinming Cao et al. «RGBD: Learning Depth-Weighted RGB Patches for RGB-D Indoor Semantic Segmentation». En: *Neurocomput.* 462 (C oct. de 2021), págs. 568-580. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2021.08.009. URL: https://doi.org/10.1016/j.neucom.2021.08.009.

[5] «Segmentación semántica para reconocimiento de escenas». En: 6.19 (2019), págs. 6-7. URL: http://doi.org/10.1109/ICIP.2014.7025197.

[6] Amin Valizadeh y Morteza Shariatee. «The Progress of Medical Image Semantic Segmentation Methods for Application in COVID-19 Detection». En: *Computational intelligence and neuroscience* 2021 (nov. de 2021), pág. 7265644. ISSN: 1687-5273. DOI: 10.1155/2021/7265644. URL: https://pubmed.ncbi.nlm.nih.gov/34840563%20https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8611358/.

[7] Bike Chen, Chen Gong y Jian Yang. «Importance-Aware Semantic Segmentation for Autonomous Vehicles». En: *IEEE Transactions on Intelligent Transportation Systems* 20.1 (2019), págs. 137-148. DOI: 10.1109/TITS.2018.2801309.

[8] Junho Jo et al. «Handwritten Text Segmentation via End-to-End Learning of Convolutional Neural Networks». En: *Multimed Tools Applications* (jun. de 2019).

[9] Mohamed Chouai, Mostefa Merah y Malika Mimi. «CH-Net: Deep adversarial autoencoders for semantic segmentation in X-ray images of cabin baggage screening at airports». En: *Journal of Transportation Security* 13 (1 2020), págs. 71-89. ISSN: 1938-775X. DOI: 10.1007/s12198-020-00211-5. URL: https://doi.org/10.1007/s12198-020-00211-5.

[10] Javier A Cardenas et al. «Intelligent Position Controller for Unmanned Aerial Vehicles (UAV) Based on Supervised Deep Learning». En: *Machines* 11.6 (2023), pág. 606.

[11] Luis G Jaimes y Juan M Calderon. «An UAV-based incentive mechanism for Crowdsensing with budget constraints». En: *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE. 2020, págs. 1-6.

[12] GA Cardona et al. «Autonomous navigation for exploration of unknown environments and collision avoidance in mobile robots using reinforcement learning». En: *2019 Southeast-Con*. IEEE. 2019, págs. 1-7.

[13] Javier Alexis Cárdenas et al. «Optimal PID ø axis Control for UAV Quadrotor based on Multi-Objective PSO». En: *IFAC-PapersOnLine* 55.14 (2022), págs. 101-106.

[14] Wilson O Quesada et al. «Leader-follower formation for UAV robot swarm based on fuzzy logic theory». En: *Artificial Intelligence and Soft Computing: 17th International Conference, ICAISC 2018, Zakopane, Poland, June 3-7, 2018, Proceedings, Part II 17*. Springer. 2018, págs. 740-751.

[15] César Antonio Toro. «Algoritmos de segmentación semántica para anotación de imágenes». En: (2019). URL: http://oa.upm.es/55407/1/CESAR_ANTONIO_ORTIZ_TORO.pdf.

[16] Olanda Prieto Ordaz y David Maloof Flores. «Segmentación Semántica para Reconocimiento de Escenas». En: *FINGUACH. Revista de Investigación Científica de la Facultad de Ingeniería de la Universidad Autónoma de Chihuahua* 6.19 (jun. de 2019), págs. 6, 7. URL: https://vocero.uach.mx/index.php/finguach/article/view/371.

[17] Alejandro Barrera et al. *Análisis, evaluación e implementación de algoritmos de segmentación semántica para su aplicación en vehículos inteligentes*. Universidad Carlos III de Madrid, sep. de 2018. URL: https://github.com/tzutalin/ros_caffe.

[18] Yu-Jin Zhang. «Image Segmentation in the Last 40 Years». En: *https://services.igi-global.com/resolvedoi/resolv 1-60566-026-4.ch286* (ene. de 2009), págs. 1818-1823. DOI: 10.4018/978-1-60566-026-4.CH286. URL: https://www.igi-global.com/chapter/image-segmentation-last-years/13824%20www.igi-global.com/chapter/image-segmentation-last-years/13824.

[19] Keith Foote. «A Brief History of Semantics - DATAVERSITY». En: (mayo de 2016). URL: https://www.dataversity.net/brief-history-semantics/.

[20] Eai Fund Official. «History of image segmentation». En: (oct. de 2018). URL: https://medium.com/@eaifundoffical/history-of-image-segmentation-655eb793559a.

[21] Dhruv Parthasarathy. «A Brief History of CNNs in Image Segmentation: From R-CNN to Mask R-CNN | by Dhruv Parthasarathy | Athelas». En: *Athelas* (abr. de 2017). URL: https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4.

[22] Xuming He, Richard S Zemel y Miguel Á Carreira-Perpiñán. «Multiscale Conditional Random Fields for Image Labeling». En: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2004), págs. 695-703.

[23] John et al. «TextonBoost: Joint Appearance, Shape and Context Modeling for Multi-class Object Recognition and Segmentation». En: *Computer Vision – ECCV 2006* (2006). Ed. por Horst, Pinz Axel Leonardis Aleš y Bischof, págs. 1-15.

[24] Torralba et al. «Context-based vision system for place and object recognition». En: *Proceedings Ninth IEEE International Conference on Computer Vision* (2003), 273-280 vol.1. DOI: 10.1109/ICCV.2003.1238354.

[25] Lijun Ren y Yuansheng Liu. «Research on the Application of Semantic Segmentation of driverless vehicles in Park Scene». En: 2020, págs. 342-345. DOI: 10.1109/ISCID51228.2020.00083.

[26] Changshuo Wang et al. «A brief survey on RGB-D semantic segmentation using deep learning». En: *Displays* 70 (2021), pág. 102080. ISSN: 0141-9382. DOI: https://doi.org/10.1016/j.displa.2021.102080. URL: https://www.sciencedirect.com/science/article/pii/S014193822100086X.

[27] Pedro Ignacio Orellana Rueda. «SEGMENTACIÓN SEMÁNTICA Y RECONOCIMIENTO DE LUGARES USANDO CARACTERÍSTICAS CNN PREENTRENADAS». Universidad de chile, 2019. URL: https://repositorio.uchile.cl/bitstream/handle/2250/173733/cf-orellana_pr.pdf?sequence=1&isAllowed=y.

[28] Derek et al. «Indoor Segmentation and Support Inference from RGBD Images». En: *Computer Vision – ECCV 2012* (2012). Ed. por Svetlana et al., págs. 746-760.

[29] Kaiming He et al. «Mask R-CNN». En: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), págs. 2980-2988. DOI: 10.1109/ICCV.2017.322.

[30] Xin Wang et al. «Path and Floor Detection in Outdoor Environments for Fall Prevention of the Visually Impaired Population». En: *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE. 2022, págs. 1-6.

[31] Gustavo A Cardona et al. «Visual victim detection and quadrotor-swarm coordination control in search and rescue environment». En: *International Journal of Electrical and Computer Engineering* 11.3 (2021), pág. 2079.

[32] Seungyong Lee, Seong-Jin Park y Ki-Sang Hong. «RDFNet: RGB-D Multi-level Residual Feature Fusion for Indoor Semantic Segmentation». En: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), págs. 4990-4999. DOI: 10.1109/ICCV.2017.533.

[33] Yang He et al. «STD2P: RGBD Semantic Segmentation Using Spatio-Temporal Data-Driven Pooling». En: (2017), págs. 7158-7167. DOI: 10.1109/CVPR.2017.757.

[34] Laura J Padilla Reyes et al. «Adaptable Recommendation System for Outfit Selection with Deep Learning Approach». En: *IFAC-PapersOnLine* 54.13 (2021), págs. 605-610.

[35] Steven Ricardo Castro Ramirez y Oscar Felipe Roncancio Avendaño. «Desarrollo de sistema de navegación para un vehículo terrestre basado en segmentación semántica para ambientes internos controlados». Universidad Piloto de Colombia, ago. de 2021. URL: http://repository.unipiloto.edu.co/handle/20.500.12277/10841.

[36] Wenbang Deng et al. «Semantic RGB-D SLAM for Rescue Robot Navigation». En: *IEEE Access* 8 (oct. de 2020), págs. 221320-221329. ISSN: 21693536. DOI: 10.1109/ACCESS.2020.3031867.

[37] Fude Cao y Qinghai Bao. «A Survey On Image Semantic Segmentation Methods With Convolutional Neural Network». En: *2020 International Conference on Communications, Information System and Computer Engineering (CISCE)*. 2020, págs. 458-462. DOI: 10.1109/CISCE50729.2020.00103.

[38] Jiafan Zhuang, Zilei Wang y Bingke Wang. «Distortion-Aware Feature Correction». En: 31.8 (2021), págs. 3128-3139.

[39] ZHIJIE LIN et al. «Image style transfer algorithm based on semantic segmentation». En: *IEEE Access* (ene. de 2021). URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9336718.

[40] Fahimeh Fooladgar y Shohreh Kasaei. «A survey on indoor RGB-D semantic segmentation: from hand-crafted features to deep convolutional neural networks». En: *Multimedia Tools and Applications* 79 (7 2020), págs. 4499-4524. ISSN: 1573-7721. DOI: 10.1007/s11042-019-7684-3. URL: https://doi.org/10.1007/s11042-019-7684-3.

[41] Alejandro de Nova Guerrero. «Detección y segmentación de objetos en imágenes panorámicas». En: ().

[42] Rui Zhang et al. «A survey on deep learning-based precise boundary recovery of semantic segmentation for images and point clouds». En: *International Journal of Applied Earth Observation and Geoinformation* 102 (2021), pág. 102411. ISSN: 1569-8432. DOI: https://doi.org/10.1016/j.jag.2021.102411. URL: https://www.sciencedirect.com/science/article/pii/S0303243421001185.

[43] Yujian Mo et al. «Review the state-of-the-art technologies of semantic segmentation based on deep learning». En: *Neurocomputing* 493 (2022), págs. 626-646. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2022.01.005. URL: https://www.sciencedirect.com/science/article/pii/S0925231222000054.

[44] Íñigo Alonso Ruiz. «Segmentación Semántica con Modelos de Deep Learning y Etiquetados No Densos». Universidad de Zaragoza, ene. de 2018. URL: https://zaguan.unizar.es/record/69800/files/TAZ-TFM-2018-012.pdf.

[45] Alberto Garcia-Garcia et al. «A Review on Deep Learning Techniques Applied to Semantic Segmentation». En: (abr. de 2017). URL: https://arxiv.org/abs/1704.06857v1.

[46] Jeremy Jordan. «An overview of semantic image segmentation.» En: (mayo de 2018). URL: https://www.jeremyjordan.me/semantic-segmentation/.

[47] Larry Hardesty. «Neural networks». En: *MIT News Office* 14 (5 oct. de 2017), págs. 503-519. ISSN: 17518520. DOI: 10.1007/S11633-017-1054-2. URL: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414.

[48] Rahul Chauhan, Kamal Kumar Ghanshala y R. C. Joshi. «Convolutional Neural Network (CNN) for Image Detection and Recognition». En: *ICSCCC 2018 - 1st International Conference on Secure Cyber Computing and Communications* (jul. de 2018), págs. 278-282. DOI: 10.1109/ICSCCC.2018.8703316.

[49] Vitaly Bushaev. «How do we 'train' neural networks». En: *Towards data science* (nov. de 2017). URL: https://towardsdatascience.com/how-do-we-train-neural-networks-edd985562b73.

[50] Arden Dertat. *Applied Deep Learning*. Nov. de 2017. URL: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2.

[51] Kevin Gurney y New York. *An introduction to neural networks*. 1997.

[52] Niklas Donges. *What Is Transfer Learning? A Guide for Deep Learning | Built In*. Sep. de 2022. URL: https://builtin.com/data-science/transfer-learning.

[53] Hyeok June Jeong, Kyeong Sik Park y Young Guk Ha. «Image Preprocessing for Efficient Training of YOLO Deep Learning Networks». En: *Proceedings - 2018 IEEE International Conference on Big Data and Smart Computing, BigComp 2018* (mayo de 2018), págs. 635-637. DOI: 10.1109/BIGCOMP.2018.00113.

[54] Rohit Kundu. *YOLO Algorithm for Object Detection Explained*. Ene. de 2023. URL: https://www.v7labs.com/blog/yolo-object-detection.

[55] Gaudenz Boesch. *YOLOv7: The Fastest Object Detection Algorithm (2023) - viso.ai*. URL: https://viso.ai/deep-learning/yolov7-guide/.

[56] Learn OpenCV. *YOLOv7 Object Detection Paper Explanation and Inference*. 2023. URL: https://learnopencv.com/yolov7-object-detection-paper-explanation-and-inference/.

[57] Zijia Yang et al. «Tea Tree Pest Detection Algorithm Based on Improved Yolov7-Tiny». En: *Agriculture* 13.5 (2023), pág. 1031. DOI: 10.xxxx/agriculture13051031. URL: https://www.mdpi.com/2077-0472/13/5/1031.

[58] Li Ma et al. «Detection and Counting of Small Target Apples under Complicated Environments by Using Improved YOLOv7-tiny». En: *Agronomy* 13.5 (2023), pág. 1419. DOI: 10.xxxx/agronomy13051419. URL: https://www.mdpi.com/2073-4395/13/5/1419.

[59] Rocío Alvarez-Cedrón García-Zarandieta. *Implementación de un modelo de detección y seguimiento de jugadores de waterpolo para el análisis de modelos de juego*. Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación. Universidad Politécnica de Madrid (UPM). 2020. URL: https://oa.upm.es/62753/.

[60] Kaiming He et al. «Mask R-CNN». En: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Oct. de 2017.

[61] Viso.AI. *Everything about Mask R-CNN: A Beginner's Guide*. 2021. URL: https://viso.ai/deep-learning/mask-r-cnn/.

[62] Jacob Solawetz. *An Introduction to the COCO Dataset*. Oct. de 2020. URL: https://blog.roboflow.com/coco-dataset/.

[63] Amazon Web Services (AWS). *Transformación de los conjuntos de datos de COCO*. Fecha de acceso. URL: https://docs.aws.amazon.com/es_es/rekognition/latest/customlabels-dg/md-transform-coco.html.

[64] Jacob Solawetz. *An Introduction to the COCO Dataset*. Oct. de 2020. URL: https://blog.roboflow.com/coco-dataset/#object-detection-with-coco.

[65] Xiaolei Wang et al. «A deep learning method for optimizing semantic segmentation accuracy of remote sensing images based on improved UNet». En: *Scientific reports* (2023), págs. 1-10.

[66] Evan Shelhamer, Jonathan Long y Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation*. 2016. arXiv: 1605.06211 [cs.CV].

[67] Liang-Chien Chen et al. «DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs». En: (mayo de 2017).

[68] Wei Liu et al. «SSD: Single Shot MultiBox Detector». En: (2016), págs. 21-37. DOI: 10.1007/978-3-319-46448-0_2.

[69] Jose León et al. «Robot swarms theory applicable to seek and rescue operation». En: *Intelligent Systems Design and Applications: 16th International Conference on Intelligent Systems Design and Applications (ISDA 2016) held in Porto, Portugal, December 16-18, 2016*. Springer. 2017, págs. 1061-1070.

[70] Gustavo A Cardona y Juan M Calderon. «Robot swarm navigation and victim detection using rendezvous consensus in search and rescue operations». En: *Applied Sciences* 9.8 (2019), pág. 1702.

[71] David Paez et al. «Distributed particle swarm optimization for multi-robot system in search and rescue operations». En: *IFAC-PapersOnLine* 54.4 (2021), págs. 1-6.

[72] Nicolás Gómez et al. «Leader-follower behavior in multi-agent systems for search and rescue based on pso approach». En: *SoutheastCon 2022*. IEEE. 2022, págs. 413-420.

[73] Edgar C Camacho, Nestor I Ospina y Juan M Calderón. «COVID-Bot: UV-C based autonomous sanitizing robotic platform for COVID-19». En: *Ifac-papersonline* 54.13 (2021), págs. 317-322.

[74] Gustavo A Cardona et al. «Robust adaptive synchronization of interconnected heterogeneous quadrotors transporting a cable-suspended load». En: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, págs. 31-37.

[75] Gustavo A Cardona et al. «Adaptive Multi-Quadrotor Control for Cooperative Transportation of a Cable-Suspended Load». En: *2021 European Control Conference (ECC)*. IEEE. 2021, págs. 696-701.