Instituto Tecnológico Autónomo de México

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA ROBÓTICO CON VISIÓN PARA DETECCIÓN DE BASURA EN AGUAS NEGRAS

TESIS

QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN MECATRÓNICA

PRESENTA

BRANDON FRANCISCO HERNÁNDEZ TRONCOSO

ASESOR

DR. EDGAR FRANCISCO ROMÁN RANGEL

CDMX. 2023

"Con fundamento en los artículos 21 y 27 de la Ley Federal de Derecho de Autor y como titular de los derechos moral y patrimonial de la obra titulada "**DISEÑO E IMPEMENTACIÓN DE UN SISTEMA ROBÓTICO CON VISIÓN PARA DETECCIÓN DE BASURA EN AGUAS NEGRAS**", otorgo de manera gratuita y permanente al Instituto Tecnológico Autónomo de México y a la biblioteca Raúl Baillères Jr., autorización para que fijen la obra en cualquier medio, incluido el electrónico, y la divulguen entre sus usuarios, profesores, estudiantes o terceras personas, sin que pueda percibir por tal divulgación una prestación."

BRANDON FRANCISCO HERNÁNDEZ TRONCOSO

_____

Fecha

_____

Firma

Agradecimientos

A mis padres, porque sin su ayuda, paciencia y amor no hubiera podido lograr todo lo que he hecho ni ser la persona que soy. Este trabajo también es de ustedes, por acompañarme en mis desveladas, por brindarme su apoyo cuando lo necesité, por escucharme cuando tuve que contarles algo, por luchar ante las adversidades, y por nunca dejar de confiar en mí. Toda la vida voy a estar agradecido con ustedes.

A mi asesor Edgar Román, porque gracias a él descubrí mi pasión e interés por la visión por computadora y el ámbito académico. Por su apoyo y paciencia en el desarollo de este trabajo, por confiar en que podíamos lograrlo, y por brindarme consejos que siempre llevaré en mente.

Al profesor Thomas, porque apesar de habernos encontrado en tiempos difíciles, me brindó el asesoramiento y ayuda que necesité para poder trabajar y lograr mis objetivos reflejados en este trabajo.

A mis amigos y amigas, porque sin ustedes la universidad no hubiera sido igual, porque me ayudaron a disfrutar y pasar momentos que guardaré por siempre, porque encontré amistades que me apoyaron y confiaron en mí a pesar de las adversidades, por haber estado conmigo y acompañarnos en este proceso.

A mis profesores, porque me ayudaron a descubrir mi vocación e interés durante mis estudios, por ayudarme cada vez que lo necesitaba, por escucharme cada vez que tenía algo en mente, por creer en mí y estar al pendiente de mi desempeño, por apoyarme cuando quise alcanzar mis objetivos.

Al ITAM, por todos los conocimientos que adquirí, porque me ayudaron a vivir experiencias únicas, porque me brindaron la ayuda necesaria para culminar mis estudios; estudiar aquí fue la mejor decisión que pude haber tomado.

RESUMEN DEL DOCUMENTO

El objetivo de este documento es presentar el proceso de diseño e implementación de un sistema robótico con visión artificial para realizar de tareas detección y clasificación de desechos plásticos submarinos mediante la navegación de ambientes acuáticos con poca iluminación. La intención de este sistema es brindar ayuda a personal de protección civil y bomberos para la identificación y clasificación de basura en zonas con presencia de inundaciones, facilitando la recolección de los desechos y liberando los puntos de drenaje para disminuir el agua acumulada.

La solución propuesta fue desarrollada mediante la implementación de redes neuronales convolucionales usando las arquitecturas YOLOv3 y YOLOv4 para el sistema de visión artificial, el modelado y manufactura con control numérico de las piezas del sistema robótico, y la integración de elementos electrónicos y periféricos que faciliten la interacción de ambos sistemas con la intención de funcionar en tiempo real. Para lograr una solución efectiva, se desarrollaron ambos sistemas de forma independiente, por lo que se describe cómo fueron generados individualmente y el proceso de integración en una solución final.

TABLA DE CONTENIDO

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

Capítulo 1

INTRODUCCIÓN

Las inundaciones son uno de los fenómenos naturales más comunes en México ocasionados por el exceso de lluvia y el desbordamiento de ríos y drenajes. Actividades humanas como la deforestación de bosques, ubicación de viviendas en zonas bajas o cercanas a ríos, y malos hábitos de desechos de basura, han facilitado la presencia de inundaciones en las ciudades [64]. El propósito de este capítulo es presentar un breve contexto del problema que se quiere resolver y una descripción del mismo. También se plantean los objetivos del proyecto a desarrollar, el alcance esperado y la metodología a seguir con la intención de generar un prototipo funcional que ayude a solucionar el problema planteado.

1.1. Contexto del problema

Históricamente en México, durante los meses de junio a octubre, ocurre el fenómeno *temporada de lluvias y huracanes* [13], trayendo consigo efectos y consecuencias en todo el país. Algunos de los daños más comunes provocados por las lluvias son afectaciones a viviendas, contaminación y desabasto de alimentos, desarrollo de virus y bacterias infecciosas, e incluso pérdida de vidas humanas [64]. Las ciudades de México, Monterrey, Guadalajara, Tijuana y Juárez son las ciudades más habitadas del país que han presentado problemas importantes relacionados con el agua, desde el abasto de agua potable hasta el desalojo de

aguas residuales y procedentes de tormentas. Por otro lado, las capitales del resto de los estados de la república han empezado a tener estos mismos problemas debido a la urbanización y falta de obras hidráulicas [12].

Algunas de las causas de las inundaciones en la Ciudad de México son la falta de saneamiento y poco mantenimiento de los sistemas de drenaje [26]. Sin embargo, más del 50 % de las inundaciones reportadas durante la temporada de lluvias son consecuencia de la acumulación de basura en las calles [1, 59], la cual es arrastrada por las corrientes de agua y provoca el bloqueo de cauces, atascamiento de coladeras y encharcamientos en vías de transporte [55]. Por otro lado, se tiene registro de que los ríos y lagos, así como las presas para la contención de aguas pluviales, tienen altos niveles de contaminación debido a los malos hábitos de desechos de basura que tiene la población mexicana [26].

Además de campañas de concientización para la población y multas federales a gente que arroje basura en la calle, se han propuesto soluciones para mitigar el efecto de las inundaciones y reducir la acumulación de agua en las calles de la ciudad. Algunas de las soluciones propuestas han sido: estudios a corto y largo plazo antes de la temporada de lluvias para pronosticar la cantidad de lluvia futura y posible reuso del agua, construcción de presas y bordos para almacenamiento de agua pluvial, uso de mejor infraestructura para el tratamiento de desechos, o el degradado y limpieza de cauces para mejorar las corrientes de agua [15]. También se han propuesto soluciones más sustentables como sistemas de captación pluvial en nuevas edificaciones o el tratamiento y reuso del agua para zonas de riego [17].

A pesar de tener diversas soluciones que se enfocan en la captación de lluvia, no se ha hecho énfasis en generar soluciones para tratar la basura que se acumula dentro y fuera de las alcantarillas. En consecuencia, resulta pertinente proponer soluciones para identificar los desechos acumulados en zonas de drenaje, facilitando las labores de bomberos y elementos de protección civil a la hora de retirar los desechos ante una inundación.

1.2. Identificación del problema

Considerando el contexto y análisis anteriores, se desea desarrollar una solución para la identificación y clasificación de basura que fluye y se acumula en puntos de desagüe, ríos y lagos de la Ciudad de México. Trabajar en estos ambientes es una tarea difícil debido a las corrientes de agua que pudieran generarse, además de contar con poca iluminación a la hora de identificar la basura presente en las denominadas *aguas negras*. Una solución a este problema ayudaría a disminuir y evitar las inundaciones a lo largo de las calles y vías de movilidad de las ciudades. También facilitaría el trabajo de elementos de protección civil y bomberos a la hora de retirar la basura en una inundación, al igual que reduciría el riesgo de contraer enfermedades o infecciones por acumulación de residuos.

1.3. Objetivo general de la solución

El objetivo de este trabajo es desarrollar un sistema robótico con la capacidad de navegar en superficies acuáticas y llevar a cabo tareas de búsqueda, identificación y clasificación de basura en ambientes con poca luz. Se busca que con este sistema se faciliten las tareas y tomas de decisiones relacionadas con la recolección de basura que realiza el personal de protección civil y bomberos con el propósito de reducir o evitar posibles inundaciones.

1.4. Objetivos específicos de la solución

Se cuenta con una serie de objetivos específicos que deben ser alcanzados durante la realización de este proyecto, donde cada uno ellos está asociado con alguno de los módulos más importantes de la solución a desarrollar. A continuación se enlistan dichos objetivos:

- Diseñar e implementar un sistema robótico capaz de navegar sobre el agua sin necesidad de ser controlado manualmente.
- Integrar un sistema capaz de tomar fotografías en tiempo real para ambientes acuáticos con baja iluminación.
- Desarrollar un sistema de visión por computadora capaz de procesar fotografías en ambientes acuáticos con poca iluminación para identificar posibles desechos plásticos y desperdicios.

1.5. Alcance de la solución

Se proponen tres ejes principales para desarrollar el proyecto, los cuales consisten en el diseño del sistema robótico, el modelo de aprendizaje de máquina para reconocimiento de basura, y las pruebas y simulaciones para verificar que los dos anteriores fueron desarrollados de manera exitosa. A continuación se enlistan y describen los alcances del proyecto considerando los ejes mencionados anteriormente:

- **Sistema robótico**: desarrollar un prototipo de sistema robótico para navegación de lagos, encharcamientos e inundaciones con presencia de basura o deshechos plásticos.
- **Sistema de visión**: generar un modelo de identificación de basura en ambientes submarinos con poca iluminación mediante técnicas de procesamiento digital de imágenes e inteligencia artificial.
- **Simulaciones en ambientes controlados**: simular diferentes ambientes de flujo y contaminación de agua, así como distintas fuentes e intensidades de iluminación.

Inicialmente se tiene considerado desarrollar ambos sistemas por separado, buscando probar cada sistema de forma independiente con la intención de evaluar si se tiene el comportamiento deseado. Después de probar cada sistema, deben ser integrados para evaluar su funcionamiento como un único componente, por lo que también se requieren pruebas y simulaciones en ambientes

controlados para identificar posibles fallas o mejoras. Una representación visual de la interacción y desarrollo de estos sistemas se muestra en la Figura 1.1.

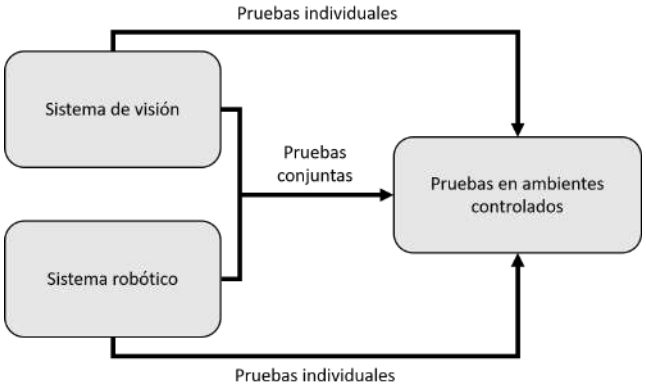


Figura 1.1: Interacción y desarrollo de los ejes principales de este proyecto.

La meta final es generar un prototipo funcional elaborado con ayuda de las máquinas por control numérico de los laboratorios del ITAM, herramientas de corte láser, y material y herramientas propias del desarrollador de este proyecto. Dicho prototipo tendrá la capacidad de navegar por las superficies de lagos, encharcamientos e inundaciones en las calles de la Ciudad de México, buscando optimizar su funcionamiento ante la presencia de ligeros flujos y corrientes de agua. El prototipo debe ser capaz de trabajar en ambientes controlados como peceras o albercas, así como en ambientes no controlados como la Barranca Tarango (presa ubicada en la alcaldía Álvaro Obregón), el lago del Arbolillo (ubicado en la alcaldía Gustavo A. Madero) o la cuenca del Río Magdalena (ubicado en la alcaldía Magdalena Contreras).

1.6. Metodología

El desarrollo de este proyecto seguirá la metodología en cascada, la cual es un proceso de desarrollo lineal que permite analizar cada parte del proyecto por etapas, desde el análisis de requerimientos hasta la implementación del protitipo [47]. El propósito de seguir esta metodología es obtener una solución de calidad

que resuelva el problema planteado en este capítulo. La Figura 1.2 muestra las etapas del proceso de desarrollo en cascada.

Figura 1.2: Modelo de proceso en cascada [47].

La elección del desarrollo en cascada se debe a la flexibilidad que proporciona para el despliegue de proyectos, adecuándola al desarrollo de cada uno de los objetivos planteados. Inicialmente se buscará identificar los requerimientos de cada objetivo para después analizar los problemas a resolver y plantear posibles soluciones al respecto. Una vez identificadas las posibles soluciones, se procederá al modelado de cada una de ellas con el fin de encontrar la solución que mejor se adecúe y resuelva el problema, lo que permitirá generar un prototipo robusto y de alta calidad. Después de modelar cada solución, se procederá a construir y generar prototipos de cada una de ellas con el propósito de verificar su funcionamiento y realizar cambios o mejoras en caso de ser necesario. Por último, se deberá realizar el despliegue de la solución para ponerla prueba en diversos ambientes con la intención de evaluar su desempeño.

1.7. Organización del documento

El presente documento se divide en 10 capítulos y 4 apéndices con la intención de describir y documentar el trabajo realizado y los resultados obtenidos con este proyecto. En este primer capítulo se incluye un breve contexto del problema a resolver y de los objetivos que se busca alcanzar, así como la metodología seguida para cumplir con un correcto desarrollo de la solución. El segundo capítulo muestra un análisis de los requerimientos necesarios para elaborar una solución al problema planteado, además de incluir las restricciones existentes y algunos trabajos relacionados con el presente proyecto. En el tercer capítulo se detallan los temas más relevantes involucrados en la realización de la solución, desde un breve contexto histórico hasta aspectos técnicos a ser implementados. El cuarto capítulo incluye una descripción exhaustiva del

funcionamineto y composición de la solución propuesta, además de una explicación de los componentes involucrados en el desarrollo de un prototipo funcional y los estándares que permiten replicar fácilmente la solución.

Siguiendo con la implementación de la solución, en el quinto capítulo se muestran las tareas relacionadas con la recolección, preparación y tratamiento de las imágenes requeridas para el desarrollo de la solución propuesta. El sexto capítulo documenta el proceso de modelado de la estructura física de la solución, comenzando con la propuesta de la estructura y su modelado computacional, y culminando con su manufactura e integración física. El séptimo capítulo explica el proceso que se siguió para integrar cada uno de los módulos desarrollados, además de mostrar algunas evidencias de la integración final antes de llevar a cabo el proceso de pruebas.

El octavo capítulo resume los resultados obtenidos con la implentación, documentando las pruebas que se realizaron y el desempeño de la solución de forma individual e integrada. El noveno capítulo provee una serie de conclusiones generadas a partir de la implementación y los resultados obtenidos, así como una definición de futuras líneas de trabajo y mejoras a la solución desarrollada. El décimo capítulo documenta algunas aportaciones que se hicieron a la comunidad científica con la intención de reducir la complejidad que tomaría desarrollar proyectos de visión por computadora, además de proponer y mejorar la implementación de un algoritmo para tratamiento de imágenes.

Por último, se incluye una serie de apéndices con algunos códigos desarrollados en Python y C++ que fueron usados para el desarrollo de este proyecto, así como los planos de cada uno de los componentes de la estructura física de la solución y el código para manufacturarlos.

Capítulo 2

ANÁLISIS DEL PROBLEMA

En este capítulo se enlistan los requerimientos funcionales del proyecto con la intención de especificar las actividades principales que la solución propuesta debe realizar. También se incluye un par de requerimientos no funcionales que permitan evaluar el desempeño de la solución bajo ciertas condiciones. Además, se enlista una serie de restricciones existentes para la elaboración de este proyecto, las cuales deben ser consideradas a lo largo de todo su desarrollo e implementación. Por último se incluye un análisis de los trabajos relacionados con este proyecto con el propósito de justificar la viabilidad del mismo.

2.1. Requerimientos funcionales

Los requerimientos funcionales son aquellas tareas y características con las que deberá contar la solución al momento de su implementación. A continuación se enlistan los requerimientos funcionales involucrados en este proyecto:

- La solución debe ser capaz de observar e identificar objetos bajo el agua sin importar la intensidad de iluminación ambiental.
- La solución debe ser sea capaz de interpretar la posición y orientación de los objetos bajo el agua en tiempo real.
- La solución debe ser capaz de identificar más de un objeto a la vez y generar la información correspondiente para cada uno de ellos.
- La solución debe transmitir información sobre la clase de los objetos detectados al usuario.
- La solución debe navegar y moverse sobre superficies acuáticas, además de obtener información ambiental para conocer la intensisdad de iluminación presente en el ambiente navegado.

2.2. Requerimientos no funcionales

Como se mencionó al inicio de este apartado, los requerimientos no funcionales son aquellas condiciones que permitirán evaluar si la solución cumple con la funcionalidad deseada. Se considera la existencia de dos requerimientos no funcionales en este proyecto, en específico para la sección de pruebas e implementación, los cuales se enlistan a continuación:

- La solución debe identificar y clasificar distintos tipos de basura y desechos que se encuentren bajo el agua.
- Es necesario que el ambiente acuático navegado se componga de al menos tres de los siguientes elementos:
  - Botellas, envases o envolturas de plástico.
  - Grasas, aceites o bebidas azucaradas.
  - Tierra o arena.
  - Polvo.

  Bajo una combinación de los elementos anteriores, la solución debe ser capaz de identificar y clasificar los desechos considerando los requerimientos funcionales descritos en la sección anterior.

2.3. Restricciones para la solución

Existen algunas restricciones que pueden retrasar o limitar la realización de este proyecto, por lo que se deben encontrar alternativas o soluciones para desarrollar la solución de manera satisfactoria. Las restricciones consideradas se enlistan a continuación:

- Se cuenta con un presupuesto limitado, debido a que el departamento de ingeniería del ITAM no puede proporcionar el material requerido para la solución. Por lo anterior, el costo del proyecto será cubierto en su totalidad por el autor de este trabajo.
- Como consecuencia del presupuesto limitado, el software a utilizar debe ser *software libre* o software proporcionado por el ITAM con la intención de reducir costos.
- El acceso a las máquinas de manufactura es limitado debido a que se debe cumplir con los protocolos de sanidad establecidos por el ITAM para trabajar en sus laboratorios de ingeniería.
- Se cuenta únicamente con una computadora para trabajar, por lo que el desarrollo de los sistemas, el modelado de la solución y la ejecución de pruebas pueden llegar a tomar más tiempo.
- Además de necesitar permisos especiales para navegar en presas, ríos o lagos protegidos por el gobierno de la Ciudad de México, las condiciones sanitarias existentes durante la realización de este proyecto limitan el acceso a lugares ideales para la realización de pruebas. En consecuencia, se debe considerar la simulación de un ambiente para que el funcionamiento de la solución pueda ser evaluado.

2.4. Trabajos relacionados

Se han presentado soluciones para diseños de robots acuáticos de bajo costo y resistentes a corrientes acuáticas, tales como el robot acuático impulsado por un rotor interno [46] y el robot pescado para monitoreo de ambientes acuáticos [40]. Sin embargo, estos diseños no consideran sistemas de visión por computadora ni cámaras para exploración, por lo que no se asegura que dichos diseños tengan el funcionamiento esperado al ser implementados para la solución del problema planteado.

También se han presentado soluciones para exploración de ambientes acuáticos con poca luz, como la propuesta por T. Van Damme para grabación de zonas arqueológicas bajo el agua [70], o la propuesta por J. Horgan y D. Toal para evaluar la calidad de fotografías tomadas bajo el agua con distintos niveles de iluminación [29]. Sin embargo, estas soluciones no contemplan la integración de sistema robótico acuático con visión que realice el procesamiento de imágenes en tiempo real o que tome decisiones para navegación e iluminación.

La solución más próxima al presente trabajo es la realizada por M. Fulton y su grupo de investigación, la cual consiste en detección de basura submarina usando distintos modelos de visión y vehículos acuáticos no tripulados [21]. Lamentablemente la publicación se centra en comparar los modelos de visión desarrollados, sin mostrar cómo fue el diseño de los vehículos acuáticos ni compartir la base de datos con la que se trabajó, por lo que resulta difícil tomar su aportación como base para este proyecto.

Capítulo 3

COMPRENSIÓN TEÓRICA

La **ingeniería**, vista como la rama de estudio que ha permitido mejorar sistemas existentes y dar solución a problemas concretos con el paso del tiempo, puede definirse como un conjunto de conocimientos y técnicas científicas aplicadas a la creación, perfeccionamiento e implementación de estructuras para la resolución de problemas [56]. Son muchas las áreas que abarca la ingeniería, sin embargo, las áreas más relevantes que están involucradas en el desarrollo de este proyecto son las áreas de **Visión por computadora**, también conocida como *Visión artificial*, y los **Vehículos acuáticos no tripulados**. A lo largo de este capítulo se da un contexto de los temas involucrados en el desarrollo de la solución propuesta, teniendo la intención de brindar más información y conocimiento que facilite el proceso del diseño, implementación y evaluación de la solución.

3.1. Visión por computadora

De acuerdo con David Marr, la visión es un proceso que permite descubrir lo que hay en el mundo y su posición dentro de él con ayuda de imágenes [5].

Tomando en cuenta esta definición, cuando se usa una cámara para simular la visión humana, y una computadora para facilitar el cálculo de operaciones matemáticas, se puede hablar de la visión por computadora como el proceso de capturar imágenes digitales con el objetivo de extraer información de ellas, además de poder transformarlas para propiciar la toma de decisiones y generar una nueva representación de la información obtenida [9]. A continuación se explican aquellos temas relacionados con el área de visión por computadora que involucran la realización de este proyecto.

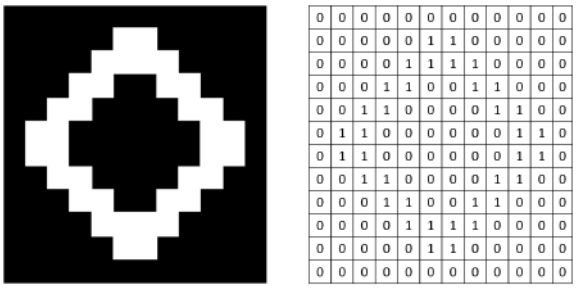
3.1.1. Contexto

Desde un punto de vista biológico, la **Visión por computadora** se puede definir como el diseño de modelos computacionales que emulan el sistema visual humano [30]. Por otro lado, desde un punto de vista ingenieril, se puede definir como el área de estudio que permite modelar sistemas autónomos que lleven a cabo tareas específicas del sistema visual humano, además de procesar la información obtenida para la toma de decisiones [30]. Tomando en cuenta las definiciones anteriores, se puede considerar que la **Visión por computadora** es el área encargada de describir el mundo que perciben las personas utilizando un conjunto de fotografías y algoritmos de procesamiento de imágenes. Para realizar esta tarea, se tiene el objetivo de reconstruir propiedades como iluminación, color y forma, o incluso obtener características deseables o patrones de las imágenes usadas [66].

El proceso de obtención de información que realizan las personas mediante la visión humana es distinto al proceso que se realiza mediante la visión artificial. En la visión humana, el cerebro divide la *señal de visión* en varios *canales*, donde cada uno de ellos tiene la tarea de obtener información específica de las imágenes observadas para poder identificar las partes más importantes, lo que permite descartar aquella información que no aporta al reconocimiento de características o a la toma de decisiones [9].

En cambio, el proceso para la obtención de información en la visión artificial requiere de una cámara y una computadora. Al inicio, la cámara captura una imagen y genera su representación mediante una *matriz de números*. Luego la imagen es proporcionada a los algoritmos de procesamiento de imágenes con la

intención de generar información de interés usando la computadora para facilitar la realización de operaciones [9]. Un ejemplo de la representación de una imagen digital es la Figura 3.1, donde se puede observar una imagen binaria de un rombo (lado izquierdo) y su respectiva representación en matriz de números (lado derecho). Por otro lado, la Figura 3.2 muestra una serie de ejemplos propuestos por R. Szeliski [66] sobre posibles aplicaciones de la visión por computadora y el procesamiento digital de imágenes.



0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	1	1	0	0	1	1	0	0	0
0	0	1	1	0	0	0	0	1	1	0	0
0	1	1	0	0	0	0	0	0	1	1	0
0	1	1	0	0	0	0	0	0	1	1	0
0	0	1	1	0	0	0	0	1	1	0	0
0	0	0	1	1	0	0	1	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Figura 3.1: Representación digital de una imagen binaria.

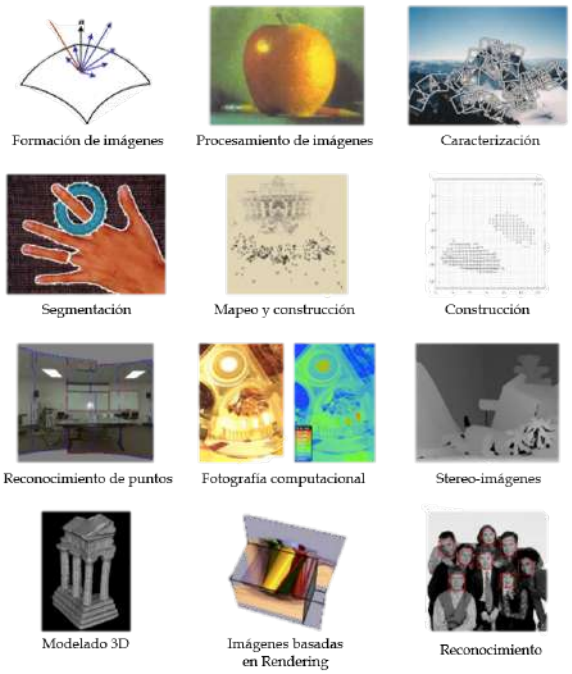


Figura 3.2: Posibles enfoques de la visión por computadora [66].

3.1.2. Aprendizaje de máquina, Aprendizaje supervisado y Redes neuronales artificiales

El **Aprendizaje de máquina** o *Machine learning* es un área que permite encontrar soluciones a problemas en áreas de visión por computadora, robótica, procesamiento de lenguaje natural, entre otras. Para encontrar estas soluciones, es necesario desarrollar algoritmos con parámetros característicos que puedan ser optimizados empleando operaciones aritméticas, herramientas estadísticas y sistemas inteligentes [2].

La intención de usar soluciones basadas en algoritmos de aprendizaje de máquina es poder trabajar con grandes volúmenes de información que puedan ser procesados por una computadora. Otra razón es que permite desarrollar algoritmos que puedan **aprender** de la información procesada, además de **optimizar** sus parámetros característicos con la finalidad de mejorar su desempeño y proveer una representación general de la información proporcionada.

Se puede considerar que el aprendizaje de máquina está compuesto por dos fases principales: la *Fase de entrenamiento* y la *Fase de prueba*. La primera de ellas consiste en optimizar los parámetros característicos del modelo con base en la información de entrenamiento. La segunda consiste en obtener resultados con la información que no fue usada durante la fase de entrenamiento para evaluar si el modelo proporciona una solución correcta al problema modelado, es decir si el problema que se desea resolver está siendo generalizado o si el modelo está únicamente memorizando la información que se usó en la etapa de entrenamiento [2]. Ambas fases pueden observarse en la Figura 3.3, las cuales son mostradas como sistemas de entrada-salida con la intención de ejemplificar el proceso realizado en cada una de ellas.

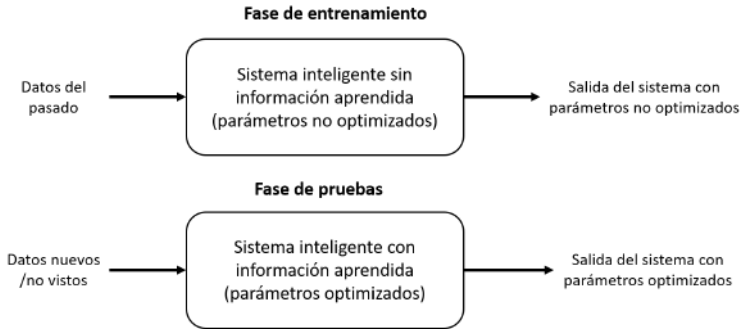


Figura 3.3: Fases para el aprendizaje de máquina.

Dentro del aprendizaje de máquina existe un paradigma conocido como **aprendizaje supervisado** o **aprendizaje con profesor**, el cual se basa en la abstracción de un **instructor** que tiene el conocimiento o la información de interés, mientras que un **aprendiz (algoritmo de aprendizaje)** desconoce dicha información y tiene el propósito de aprenderla [27]. Para que el aprendiz pueda generalizar una solución, buscará **ajustar sus parámetros característicos** de acuerdo con la información proporcionada y una retroalimentación obtenida a partir de los aciertos y errores que cometa, permitiendo otorgar respuestas correctas ante información desconocida después de la fase de entrenamiento [27].

Una representación visual del aprendizaje supervisado se muestra en la Figura 3.4, la cual abstrae al paradigma de aprendizaje con profesor como un *sistema de entrada-salida*. De esta manera, para cierta información a la entrada, el algoritmo de aprendizaje proporciona un valor de respuesta a la salida que es **comparado** con el valor **deseado** proporcionado por el *profesor*. La ventaja de este paradigma es la facilidad que otorga para optimizar los parámetros característicos del algoritmo de aprendizaje, llevándolo a cabo mediante una estrategia de retroalimentación o corrección del error [27].

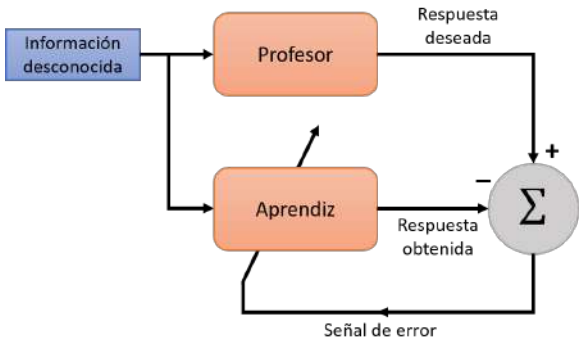


Figura 3.4: Aprendizaje supervisado visto como un sistema de entrada-salida.

Concretamente, el aprendizaje supervisado permite modelar sistemas que establezcan relaciones de entrada y salida de información con base en ejemplos proporcionados para su aprendizaje. Durante la fase de entrenamiento, el algoritmo de aprendizaje es alimentado con parejas de valores de entrada y salida, permitiendo evaluar y corregir el funcionamiento del algoritmo, así como optimizar sus parámetros característicos de acuerdo con las respuestas obtenidas a la salida. Luego, durante la fase de pruebas, se usan los valores calculados para los parámetros característicos con la intención de obtener resultados ante nueva información desconocida, permitiendo evaluar el desempeño del algoritmo con dichos parámetros, y realizar mejoras en caso de ser necesario [39].

Un algoritmo común en el aprendizaje supervisado son las **Redes neuronales artificiales**, las cuales pueden definirse como un algoritmo compuesto por unidades simples de procesamiento que, de forma paralela y distribuida, procesan cantidades masivas de información con la intención de almacenar conocimiento experimental y hacerlo disponible para las personas [27]. El objetivo de desarrollar redes neuronales artificiales es emular el funcionamiento del cerebro como elemento de aprendizaje, el cual puede verse como un procesador de información compuesto de múltiples unidades simples de procesamiento llamadas *neuronas*. Todas las neuronas que componen al cerebro operan de forma paralela, además de estar conectadas entre sí por medio de las *sinapsis* [2], por lo que se se puede notar el propósito que buscan emular las redes neuronales artificiales y su composición.

El modelo de una red neuronal artificial como un algoritmo de aprendizaje computacional inicia con la creación de las unidades simples de procesamiento, también llamadas **perceptrones**. Los perceptrones cuentan con *conexiones de entrada* que permiten emular las sinapsis del cerebro, además de ser las encargadas de conectar las neuronas entre sí. También se compone de **pesos de aprendizaje** asociados a cada entrada, los cuales son los parámetros caracterísiticos de este algoritmo de aprendizaje y ayudan a definir la fuerza con la que la neurona aprenderá la información recibida. El tercer elemento de los perceptrones es el **sumador**, el cual tiene la tarea de realizar la suma aritmética de la multiplicación de cada valor de entrada recibido con su respectivo peso de aprendizaje. El cuarto elemento del perceptrón es la polarización o **bias**, que tiene la intención de incrementar o decrementar el acumulado del sumador y agregar variabilidad. Todos los elementos anteriores en conjunto permiten generar un resultado a la salida que será proporcionado a otra neurona para su aprendizaje.

Los perceptrones también pueden entenderse como un sistema que proporciona una respuesta a la salida ante múltiples fuentes de información a su entrada. La Figura 3.5 tiene el propósito de ejemplificar visualmente la representación de un perceptrón como un sistema de entrada-salida.

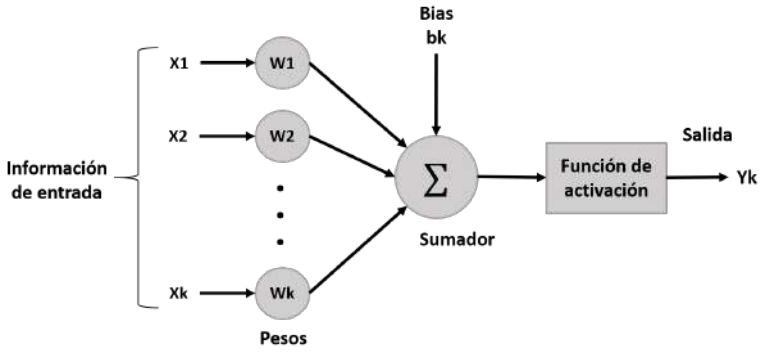


Figura 3.5: Modelo del perceptrón.

Tomando en cuenta los elementos descritos anteriormente, y considerando que para $\boldsymbol{k}$ fuentes de información a la entrada del perceptrón se cuenta con $\boldsymbol{k}$ pesos correspondientes a cada fuente y una polarización, el resultado a la

salida del sumador está dado por la siguiente operación aritmética:

$$S = w_1 * x_1 + w_2 * x_2 + ... + w_n * x_n + b_k.$$

Debido a que los valores obtenidos a la salida del sumador pueden ser muy grandes y variados, resulta pertienente limitar la amplitud de los valores de salida de la neurona y escalar el resultado del sumador a un rango menos variable, por lo que el perceptrón cuenta con un último elemento llamado **función de activación** que cumple con estos requerimientos [27]. La interacción de todos los elementos anteriores facilitan la resolución de problemas desde un enfoque no lineal, permitiendo construir estructuras de redes neuronales profundas para modelar problemas que con una solución lineal no sería suficiente. Un ejemplo de arquitectura de una red neuronal artificial puede observarse en la Figura 3.6, la cual se compone de múltiples capas o *layers* de perceptrones conectadas entre sí.

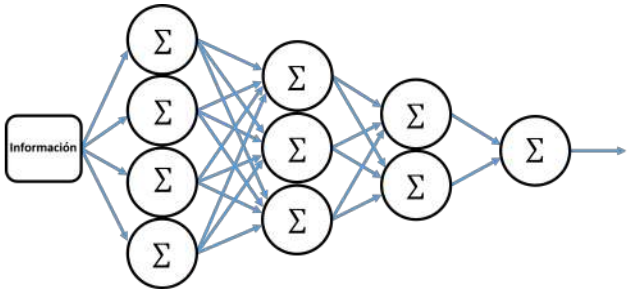


Figura 3.6: Ejemplo de arquitectura de red neuronal.

Para que las redes neuronales aprendan y tengan el desempeño deseado, se requiere ajustar los pesos de entrada de cada neurona con base en los errores cometidos. Para ello, es necesario implementar un *algoritmo de propagación hacia atrás* que proporcione a las neuronas anteriores información sobre el resultado obtenido a la salida, lo que permite modificar los valores de dichos pesos de acuerdo con la comparación entre el valor esperado a la salida y el resultado obtenido [27].

3.1.3. Redes neuronales convolucionales

También conocidas como *CNN* por sus siglas en inglés, las **Redes neuronales convolucionales** son un tipo de redes neuronales multicapa diseñadas para reconocer formas y patrones en información compuesta por dos o más dimensiones. La ventaja de usar este tipo de redes neuronales es la facilidad de poder trabajar con información caracterizada por altos niveles de distorción, por ejemplo alta varianza o problemas de escalamiento o sesgo [27].

De acuerdo con Y. LeCun, las redes neuronales convolucionales pueden definirse como simples redes neuronales que usan la operación de **convolución** en vez de realizar las operaciones aritméticas comunes en al menos una de sus capas [38]. Para llevar a cabo estas operaciones de convolución, se requiere de dos elementos principales: una **imagen de entrada**, y un ***kernel*** o ***filtro***, de tal forma que el kernel sea el operador que modifique la imagen de entrada a la hora de realizar la convolución, permitiendo obtener caracterísitcas o patrones de las imágenes que puedan ser aprendidos por los perceptrones. LeCun y Bengio proponen tres operaciones elementales en las redes para el reconocimiento de patrones [37], las cuales se enlistan a continuación:

- **Extracción de características:** Cada neurona toma a su entrada la información generada por las neuronas de la capa anterior. Luego mediante la operación de convolución se extraen las características locales de la imagen de acuerdo con el kernel usado.
- **Mapeo de características:** Con la intención de reducir la cantidad de parámetros característicos a optimizar, las neuronas de la red son forzadas a transmitir los valores o *pesos* de cada kernel usado para cada mapa de características. Los mapas de características son una forma de representar bidimensionalmente la información que se va obteniendo como resultado de las operaciones de convolución, por lo que cada capa de la red está compuesta por múltiples mapas de características.
- **Sub-muestreo o *subsampling*:** Después de las operaciones efectuadas en una capa de convolución, se puede realizar un sub-muestreo de los mapas de características obtenidos, permitiendo reducir el tamaño de los mapas y disminuyendo la sensibilidad ante cambios o distorciones.

Un ejemplo de aplicación de las operaciones elementales anteriores puede observarse en la Figura 3.7. El orden de las operaciones aplicadas se ennumera a continuación:

1. Convolución a la imagen de entrada con 4 *kernels* de tamaño 5x5.
2. Subsampling para reducir de tamaño cada mapa de características a la mitad de su tamaño original.
3. Convolución a los mapas de salida del subsampling con 12 kernels de tamaño 5x5x4.
4. Subsampling para reducir el tamaño de cada mapa a la mitad de su tamaño de salida.
5. Convolución a los mapas de salida del subsampling anterior con 26 kernels de tamaño 4x4x12.

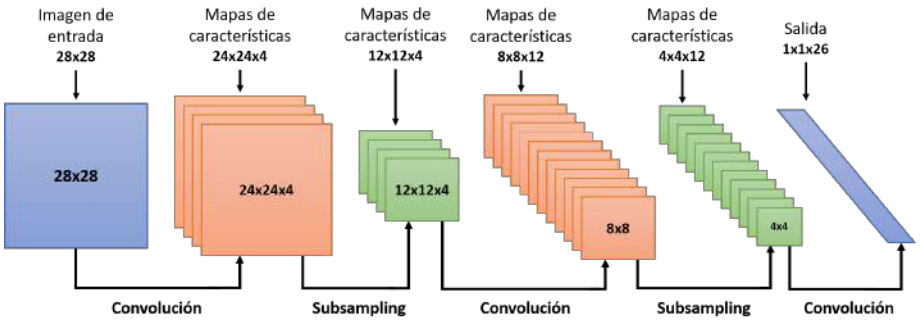


Figura 3.7: Procesamiento de imágenes en CNN [37].

3.1.4. Métricas de evaluación

Una tarea importante al construir un modelo de aprendizaje de máquina es realizar una correcta evaluación de su desempeño, lo cual permite identificar si el modelo seleccionado cumple con las expectativas y restricciones planteadadas en la definición del problema, o si debiera seleccionarse alguna otra alternativa [50]. A pesar de que existe una gran variedad de métricas para evaluar los desempeños de los algoritmos, no hay un concenso sobre qué métrica es la ideal para valorar algún algoritmo en particular, por lo que resulta necesario tener que elegir más de una métrica para tener una mayor idea del comportamiento del modelo [60].

En el caso de modelos de clasificación, existen algunas métricas clave que proporcionan una idea de qué tan bueno es el modelo después del proceso de entrenamiento. Para estas métricas es necesario considerar las definiciones [23] siguientes:

- **Verdaderos positivos (TP)**: Elementos que **pertenecen** al grupo de interés y que fueron **identificados correctamente**.
- **Falsos positivos (FP)**: Elementos que **no pertenecen** al grupo de interés y que fueron **asociados incorrectamente** al grupo.
- **Verdaderos negativos (TN)**: Elementos que **no pertenecen** al grupo de interés y que fueron **asociados correctamente** a otro grupo.
- **Falsos negativos (FN)**: Elementos que **pertenecen** al grupo de interés y que fueron **asociados incorrectamente** a otro grupo.

A partir de las anteriores definiciones, se pueden introducir las siguientes métricas de evaluación [18] [60] para modelos de aprendizaje de máquina destinados a resolver problemas de clasificación:

- **Precisión**: Esta métrica evalúa la proporción de asociaciones correctas a un grupo de interés sobre todas las asociaciones realizadas a dicho grupo por el modelo. En otras palabras, la precisión (*precision* en inglés) puede calcularse de la siguiente forma:

$$Precision = TP/(TP + FP).$$

- **Sensibilidad**: Esta métrica evalúa la proporción de asociaciones correctas a un grupo de interés sobre todas las asociaciones que debieron ser realizadas a dicho grupo por el modelo. En otras palabras, la sensibilidad (*recall* en inglés) puede calcularse de la siguiente forma:

$$Recall = TP/(TP + FN).$$

- **F1**: Esta métrica combina tanto la precisión como la sensibilidad para reportar un promedio de ambas evaluaciones, lo que permite identificar qué tan bueno es el modelo en primera instancia si se desea una evaluación general. La métrica F1 (también conocida como *F1-score* en inglés) puede calcularse de la siguiente forma:

$$F1 - score = (2 * Precision * Recall)/(Precision + Recall).$$

- **Exactitud**: Esta métrica evalúa la proporción de asociaciones correctas que realizó el modelo sobre todas las asociaciones realizadas, por lo que la exactitud (*accuracy* en inglés) se puede calcular de la siguiente forma:

$$Accuracy = (TP + TN)/(TP + FP + TN + FN).$$

3.1.5. Visión por computadora bajo el agua

La **Visión por computadora bajo el agua** es un área de reciente interés ya que representa un ambiente no ideal para la obtención óptica de imágenes. El procesamiento de imágenes capturadas bajo el agua involucra problemas como dispersión y atenuación de la luz, provocando una distorsión en la visión de la cámara y afectando la calidad de la información a obtener de las imágenes capturadas [29].

Existen varios retos al momento de trabajar en tareas de visión por computadora bajo el agua, sin embargo, los retos con mayor impacto son la dificultad de encontrar características relevantes en las imágenes, así como la ausencia de luz natural que provea mayor visibilidad de los objetos [41]. La ausencia de luz natural puede mitigarse con la presencia de luz artificial, de tal manera que cuando la luz natural sea absorbida al propagarse a través del agua, sea la luz

artificial la que ofrezca la posibilidad de extender el rango de visión y mejorar la claridad de las imágenes capturadas [58].

Unir técnicas no comunes para el procesamiento de imágenes con algoritmos de aprendizaje de máquina facilita las tareas de detección y clasificación de objetos bajo el agua. Por lo tanto, se ha considerado el uso de las operaciones y modelos descritos anteriormente, además de algunas técnicas de generación de información, con la intención de obtener buenos resultados a la hora de la exploración e identificación de objetos.

3.2. Vehículos acuáticos no tripulados

Un elemento a destacar para la exploración de ambientes acuáticos es el desarrollo de **Vehículos acuáticos no tripulados** o *Unmanned underwater vehicles*, los cuales facilitan la realización de actividades como topografía de ríos y mares [74], inspección y reparación de estructuras bajo el agua [35], localización e inspección [19], entre muchas otras.

También conocidos como *UUV* por sus siglas en inglés, los **vehículos acuáticos no tripulados** son todos aquellos robots operados con mínima o nula intervención humana para tareas de navegación sobre la superficie acuática o debajo de ella [10]. Algunos ejemplos de UUV pueden observarse en las Figuras 3.8 y 3.9, los cuales son propuestas de diseño de vehículos para tareas de exploración.

Figura 3.8: Hidroavión diseñado por DRS Technologies, Inc. [73].

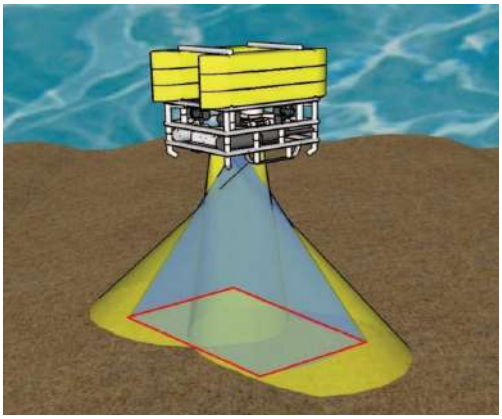


Figura 3.9: Diseño de UUV para navegación submarina [29].

Por otro lado, los UUV requieren de funcionalidades específicas para su desempeño óptimo [71], dentro de las cuales destacan:

- Registro y emparejamiento entre la información proporcionada por los sensores y la información de navegación.
- Cierto nivel de autonomía.
- Conocimiento parcial o total de la posición del vehículo en el ambiente.

Específicamente para tareas de visión, es común que los UUV cuenten con cámaras que proporcionen información de interés según sea la aplicación deseada, sin embargo, también se requiere de un digitalizador de videos en caso de realizar tareas sobre procesamiento de video, además de sensores de navegación y una fuente de iluminación externa que proporcione mayor visión en el ambiente que se esté navegando [29]. La unión de todos estos elementos ofrece una alternativa para vencer los retos de trabajar en soluciones de visión por computadora bajo el agua, así como contar con algunas ventajas [29] como:

- Fácil interpretación de la información óptica recibida por los humanos, permitiendo una mejor interacción humano-máquina para la optimización del desempeño del vehículo acuático y la captura de imágenes.

- Bajo costo para el desarrollo de sistemas ópticos, donde los únicos recursos necesarios son una cámara, un procesador de imágenes y algún elemento que proporcione luz externa.
- Generación de información espacial como posición en tres dimensiones (estereovisión) y orientación.

La integración de un sistema robótico para navegación de aguas, en conjunto con un sistema de visión y procesamiento de información ambiental, ofrece soluciones a diversos problemas de exploración e identificación de objetos, por lo que el desarrollo de un vehículo acuático no tripulado para la tarea de detección y clasificación de basura es una opción viable, además de ser una solución baja en costo y aplicable a más problemas de navegación.

Capítulo 4

DISEÑO GENERAL DE LA SOLUCIÓN

En este capítulo se incluye una serie de soluciones alternativas al problema presentado en el apartado *Análisis del problema*, permitiendo analizar sus respectivas desventajas y justificar la realización de la solución propuesta. También se incluye una visión general de la solución con la intención de mostrar su división en módulos, además de exponer la forma en que dichos módulos trabajan conjuntamente para generar un prototipo funcional. Adicionalmente se incluye una explicación más detallada de la solución con la intención de mostrar las tecnologías, diseños y componentes usados para su implementación, así como una descripción de los estándares utilizados con el propósito de que la solución esté apegada a las normas establecidas por organismos reguladores, además de ser compatible con herramientas usadas internacionalmente y que pueda ser replicada fácilmente.

4.1. Alternativas de solución

Existen algunas soluciones alternativas que ayudarían a reducir el problema de acumulamiento de deshechos en ambientes acuáticos y calles de las ciudades, además de que facilitarían el flujo de agua en puntos de desagüe ante la presencia de objetos que lo impidan. Por otro lado, hay soluciones que permitirían la identificación y clasificación de basura en puntos de interés, funcionando como una herramienta de apoyo a la hora de realizar tareas de exploración, recolección de desechos y limpieza. Lamentablemente, todas las soluciones alternativas que fueron consideradas presentan limitaciones que no las hacen ser completamente viables para solucionar el problema expuesto. Ante ello, es necesario pensar en una solución integral que pueda ser una herramienta de ayuda para elementos de protección civil y bomberos en tareas de detección y clasificación de basura.

La **primera solución alternativa** considerada es la implementación de trituradores de basura en las coladeras y puntos de desagüe de las ciudades. El objetivo de estos trituradores es que toda la basura que llegue y se acumule en ellos pueda ser molida y mezclada con la corriente de agua, permitiendo un flujo constante de la misma y facilitando su tratamiento posterior.

Una de las desventajas de esta alternativa es el mantenimiento a los sistemas de trituración, ya que pueden ser tareas riesgosas para la persona encargada de llevarlas a cabo. Otra desventaja es que los peatones podrían resultar accidentados ante cualquier descuido o atrapamiento con este sistema, por lo que se requeriría seguir una estrategia compleja para que su implementación sea segura para los peatones. La tercera desventaja recae en la cantidad de coladeras que existen y el costo de implementar estos sistemas en cada una de ellas. Si solo se considera una implementación en la Ciudad de México, se tendrían que implementar alrededor de 250,000 sistemas como este [42], por lo que resulta un proyecto poco costeable tanto local como nacionalmente. La última desventaja es que esta solución aborda únicamente los problemas de acumulación de basura en coladeras y puntos de desagüe, sin ayudar a solucionar el problema del acumulamiento de basura en ríos o lagos. Implementar una solución como esta requiere realizar estudios sobre los puntos donde más se acumula la basura, lo cual tomaría mucho tiempo para llevarlo a cabo e incurriría en costos mayores.

La **segunda solución alternativa** considerada es la implementación de un sistema robótico submarino que pueda identificar deshechos plásticos en tiempo real, además de poder clasificarlos de acuerdo con el tipo de plástico del cuál estén hechos (PET, PVC, PE-HD, entre otros). Una solución de este tipo permitiría que el mismo sistema robótico libere las coladeras y puntos de desagüe, reduciendo el riesgo al que se exponen los personales de protección civil y bomberos ante tareas que involucran corrientes de agua y espacios insalubres.

A pesar de ser una solución viable, el sistema robótico debe ser capaz de sumergirse y soportar la presión al estar navegando debajo de la superficie acuática, además de ser a prueba de filtraciones de agua y resistente ante las fuertes corrientes. Desafortunadamente los requerimientos anteriores no se pueden asegurar debido a la restricción del material disponible para realizar la solución. Otra limitante es el tiempo con el que se cuenta para entregar un prototipo funcional ya que se requiere realizar pruebas en ambientes reales durante la temporada de lluvias, lo que limita las oportunidades de probar la solución. Una limitante más es que el desarrollo de un sistema robótico submarino que sea capaz de liberar la basura implica un diseño más complejo, compuesto de subsistemas mecánicos capaces de tomar la basura y almacenarla en algún lado, requiriendo más tiempo para generar un prototipo funcional y materiales que no se tienen disponibles.

La **tercera solución alternativa** considerada es la implementación de un sistema de *Internet de las Cosas* en las coladeras y puntos de drenaje de las ciudades. La intención de desarrollar un sistema inteligente es identificar la presencia y acumulación de basura en puntos estratégicos, además de identificar incrementos en los niveles de agua en las calles. Implementar esta solución facilitaría la toma de decisiones pertinentes con antelación sin necesidad de esperar a que la lluvia termine, o incluso sin esperar a que cambien drásticamente los niveles de agua acumulada.

La principal desventaja de esta solución es que únicamente considera el envío de señales y comunicación mediante Internet y sensores, por lo que se especializa en el área de telecomunicaciones sin contemplar algún sistema mecánico o alguna implementación relacionada con mecatrónica. Otra desventaja es que, incluso aunque se tengan umbrales muy bajos para determinar si hay acumulación de basura o aumento en los niveles de agua, no se puede asegurar que las acciones

que tomen las autoridades correspondientes sean suficientes para evitar que se generen inundaciones. Otra limitante está relacionada con los costos necesarios para su implementación, de tal manera que escalarlo a nivel ciudad o nacional requeriría una gran inversión. Por último, esta solución tampoco atacaría los problemas de acumulación de basura en ríos y lagos, por lo que resulta una solución incompleta.

Debido a que no se puede asegurar una solución alternativa con un bajo costo o que se ajuste a las restricciones descritas en el apartado *Análisis del problema*, a continuación se describe la solución propuesta al problema planteado en este trabajo, la cual está pensada en generar un prototipo funcional con un largo tiempo de vida, de fácil uso y que sea de sencilla replicabilidad.

4.2. Descripción del diseño de la solución

La solución que se presenta en este apartado tiene la intención ser un sistema robótico (de ahora en adelante *robot*), capaz de observar y reconocer distintos tipos de deshechos plásticos presentes en coladeras, puntos de desagüe, ríos y lagos. También se tiene la intención de que sea un sistema capaz de navegar sobre las superficies acuáticas mencionadas anteriormente, además de que pueda clasificar la basura que se presente mientras navega y que sea un asistente a elementos de protección civil y bomberos en la identificación de los objetos a la hora de liberar o recolectar la basura acumulada.

A continuación se muestra la Figura 4.1, la cual es una representación visual del sistema propuesto en este proyecto. Adicionalmente, cada uno de los modulos que componen a la solución son explicados con la intención de especificar las tareas y objetivos que se debieran desarrollar y cumplir.

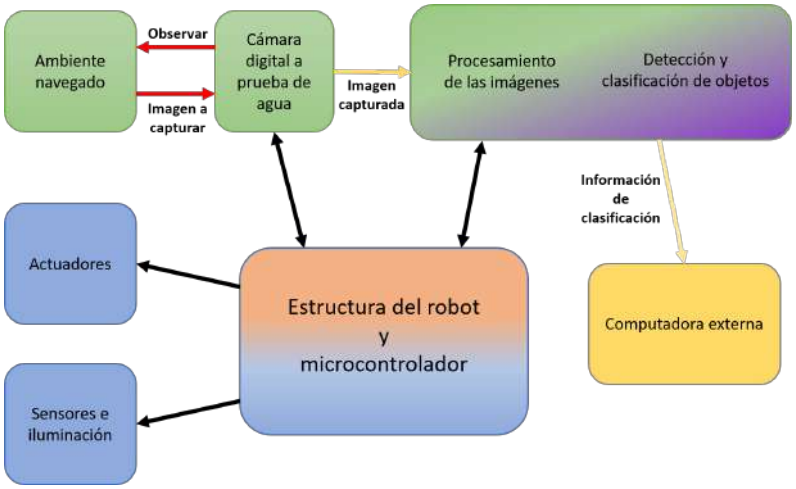


Figura 4.1: Diagrama general de la solución.

- **Estructura del robot:** es el módulo físico sobre el cual están montados todos los componentes periféricos, actuadores y unidades de procesamiento necesarios para desarrollar la solución propuesta. Este módulo está a cargo de la **estructura y diseño del robot**, además de la distribución de los componentes electrónicos, con el objetivo de que la estructura pueda navegar sin complicaciones sobre los ambientes acuáticos de interés.

  En la Figura 4.1 se puede observar un bloque central llamado *Estructura del robot y microcontrolador*, donde este módulo de la solución corresponde únicamente a la parte de la *Estructura del robot* asociado al color naranja del bloque.

- **Visión**: este módulo está dividido en dos submódulos con el propósito de explicar detalladamente el proceso de reconocimiento de los objetos e identificar las tareas requeridas:

  - El primer submódulo corresponde a la **obtención y procesamiento de las imágenes**. Considerando que uno de los requerimientos del sistema es caracterizar los objetos en tiempo real, es necesario que se **obtengan imágenes de manera constante**, las cuales recibirán un procesamiento previo antes de ser alimentadas al sistema que evalúe y detecte la presencia de basura en cada imagen generada.

En la Figura 4.1, los bloques *Ambiente navegado* y *Cámara digital* son los encargados de **capturar las imágenes** que serán evaluadas para indicar la presencia de basura o desechos, por lo que están señalados con un mismo color verde.

- El segundo submódulo corresponde al sistema de **evaluación de presencia y detección de basura** (de ahora en adelante **sistema de detección**). Para este submódulo es requerido diseñar un algoritmo, o adaptar algoritmos existentes, que sea capaz de **identificar la presencia** de basura submarina en cada imagen que se le proporcione, además de poder clasificar la basura identificada de acuerdo con los tipos de basura definidos en el algoritmo.

Es importante mencionar que el procesamiento a realizar para cada imagen capturada se debe realizar usando una **microcomputadora (unidad de procesamiento)**, la cual ejecte constantemente el algoritmo desarrollado para procesar cada imagen recibida. Otro propósito que tiene la microcomputadora es ejecutar el sistema de detección de basura para cada imagen post-procesada, tratando de cumplir con el objetivo de detección de objetos en tiempo real.

Teniendo en cuenta la información anterior sobre el módulo de visión, en la Figura 4.1 se puede observar un bloque conformado por el *Procesamiento de las imágenes* y la *Detección y clasificación de objetos*. A pesar de que estas tareas se realizan de forma independiente, ambas se agrupan en el mismo bloque debido a que sus algoritmos correspondientes se ejecutan en la misma microcomputadora. De esta manera, el procesamiento de las imágenes está asociado al color verde del bloque por pertenecer al submódulo *Obtención y procesamiento de las imágenes*, mientras que el reconocimiento de los objetos y su clasificación está asociado al color morado por pertenecer al submódulo *Sistema de detección*.

- **Movimiento y navegación del sistema**: similar al módulo de Visión, este módulo está dividido en submódulos con la intención de identificar las tareas que lo componen y la forma en cómo interactúan:

- El primer submódulo corresponde a los **Actuadores** del robot. Este submódulo es el encargado de **ejecutar los movimientos** para que la estructura pueda navegar sobre la superficie acuática.
- El segundo submódulo corresponde a **Sensores e iluminación** del robot. Este submódulo es el encargado de **generar información acerca del ambiente** en el cual se está navegando, además de evaluar si la iluminación proporcionada es suficiente para un buen desempeño del sistema de reconocimiento de basura.
- Para que los submódulos anteriores funcionen de manera óptima se requiere de un **Microcontrolador**, el cual debe proporcionar la señales necesarias para indicar a los *Actuadores* sobre los acciones necesarias para navegar. También es el encargado de recibir y procesar la información proporcionada por los *Sensores*, así como ofrecer la iluminación necesaria para asegurar más visibilidad a la hora de capturar fotografías de los objetos.

Se puede observar en la Figura 4.1 que este módulo está asociado a los bloques de color azul debido a que pertenecen al mismo conjunto de componentes electrónicos del sistema, por lo que deben trabajar en conjunto para un correcto funcionamiento y una buena navegación del robot. La inteción de incluir en el mismo bloque al microcontrolador junto con la *Estructura del robot* es poder representarlo como la parte central de los subsistemas mecánico y eléctrico.

- **Comunicación**: este módulo tiene dos funciones principales: **transmitir las imágenes capturadas** hacia la unidad de procesamiento, y **mostrar los resultados de la clasificación de basura** en una pantalla externa. Por un lado, la transmisión de las imágenes capturadas es necesaria para procesarlas y ejecutar el algoritmo de reconocimiento y clasificación de objetos sobre cada una de ellas. Por otro lado, la visualización de resultados en una pantalla externa es necesaria para monitorear el comportamiento del robot y observar si efectivamente se están obteniendo resultados correctos, o en su defecto evaluar si deben ser modificadas su operación o navegación.

En la Figura 4.1 se puede observar que el bloque *Computadora externa* y las flechas *Imagen capturada* e *Información de clasificación* están asociados al color amarillo, el cual fue usado para mostrar visualmente el proceso de comunicación durante la ejecución de las tareas del robot.

4.3. Módulo de visión

De acuerdo con la secuencia de funcionamiento de la solución propuesta, el diseño del módulo de visión será explicado considerando los bloques que componen los submódulos presentados en la sección anterior. De esta forma, se busca justificar la elección de los componentes necesarios para que el sistema de visión cumpla con los requerimientos funcionales descritos en el apartado *Análisis del problema*, así como discutir posibles alternativas de dichos componentes frente a las opciones elegidas.

Considerando el diagrama general de la solución de la Figura 4.1, el primer bloque a explicar del sistema es el *Ambiente navegado*, el cual es parte del submódulo **obtención y procesamiento de las imágenes**. El propósito de este bloque es cumplir con las restricciones para probar la solución desarrollada, además de poseer las características requeridas para capturar imágenes y evaluar el desempeño del modelo de detección de objetos.

Tomando en cuenta el alcance de la solución que fue presentado en el apartado de *Introducción*, el robot debe ser capaz de navegar sobre cualquier superficie acuática como ríos, lagos, encharcamientos e inundaciones que cumplan con los requerimientos no funcionales presentados en el apartado *Análsis del problema*, por lo que los ambientes navegados deben contener objetos plásticos sumergidos, al igual que compuestos químicos (aceites, detergentes o grasas) y tierra o polvo que impidan la visibilidad de la cámara o compliquen la captura de imágenes de los objetos. Adicionalmente, dado que el objetivo de este proyecto es la detección y clasificación de objetos en aguas negras, es necesario que la luz ambiental sea reducida para que la iluminación pueda ser proporcionada de forma externa.

Otro bloque a considerar es la *Cámara digital* con la que se capturan las imágenes en el ambiente navegado, la cual es el componente encargado de ofrecer la información de interés al modelo de detección de objetos. Debido a que la cámara es un componente crítico en el funcionamiento del robot, se espera que las imágenes capturadas posean la mayor resolución posible, además de una alta velocidad de captura y fácil adaptación a la estructura física del robot. Teniendo en mente lo anterior, se analizaron distintas opciones de cámaras considerando el presupuesto limitado:

- Cámara APEMAN A79: Esta cámara se caracteriza por su bajo precio de venta, por su compatibilidad con accesorios, una sumergibilidad de hasta 40 metros con carcasa y alta resolución para tomar fotografías y grabar video en 4K [3]. Su principal desventaja es el tiempo que toma importarla de su país de origen debido a que no tiene un distibuidor en México, por lo que tomaría más tiempo el desarrollo de la solución.
- Sony DSC-RX0M2G: Esta cámara destaca su diseño *ultra compacto*, lo que le permite ser montada o guardada en cualquier parte sin necesidad de ocupar mucho espacio. Otras de sus características son la capacidad de sumergirse hasta 10 metros sin necesidad de una carcasa protectora y su alta resolución en 4K para video. La desventaja de esta cámara recae en su costo, ya que tiene un precio de venta al público demasiado alto y resulta inviable de acuerdo con las restricciones de presupuesto [65].
- Victure AC700: Es una cámara que cuenta con características similares a la APEMAN, donde su principal atributo es su bajo precio de venta. Permite una sumergibilidad de hasta 40 metros con carcasa protectora y una alta compatibilidad con accesorios para diversas actividades deportivas. Sin embargo, su calidad fotográfica y de video es menor en comparación con las opciones anteriores, por lo que no permitiría un funcionamiento óptimo considerando un ambiente como las aguas negras [25].

Si bien las características y propiedades de las cámaras enlistadas anteriormente parecieran ser suficientes para cumplir con los requerimientos del sistema, se debe considerar que agregar una cámara al robot implica aumentar el peso de la estructura completa. También, agregar una cámara externa implica modificar

algunas variables físicas como su centro de masa y su centro de gravedad, por lo que la selección de una cámara externa debe estar basada en el diseño final del robot y la distribución correcta de todos los componentes.

Tomando en cuenta el diseño del robot a desarrollar (explicado en la sección *Diseño de la estructura del robot*), así como la distribución de los componentes internos y la forma de trabajo del modelo de detección de objetos, se tomó la decisión de trabajar con una cámara interna que cumpla con los requerimientos mencionados y que sea de fácil acceso. Las cámaras elegidas para trabajar de forma individual se enlistan a continuación:

- Cámara GoPro Hero 7 Black: Esta cámara se caracteriza por su diseño compacto y un peso de 117 gramos, teniendo poca repercusión en el peso o en el diseño final del robot. Otras características que ofrece son las distintas calidades fotográficas según sea el ambiente en el que se esté utilizando, además de mejorar la calidad de las fotografías y videos al incorporar un autoestabilizador de imagen. Por último, la cámara posee un GPS que permitirá encontrar el robot en caso de extravío durante la operación [57]. Es importante destacar que esta cámara tiene un costo de venta medio, por lo que puede ser adquirida respetando la restricción presupuestal.
- Celular Xiaomi Mi 9T (cámara trasera integrada): Este celular destaca por su diseño delgado y su peso de 190 gramos, lo que lo hace una buena elección sin repercutir tanto en el peso o en el diseño final del robot. Cuenta con una cámara integrada de 13 MP y un procesador Snapdragon 730, permitiendo una captura de fotos veloz y un procesamiento interno compatible con el envío de imágenes en tiempo real [45]. La elección de este componente se debe a que es un dispositivo con el que se cuenta físicamente, por lo que se ajusta al presupuesto limitado al no implicar un gasto.

Figura 4.2: Cámara GoPro Hero 7 Black a usar.

Figura 4.3: Celular Xiaomi Mi 9T a usar.

El siguiente bloque de este mismo submódulo es el *Procesamiento de las imágenes*, para el cual es necesario considerar la computadora que realice el tratamiento las imágenes y que ejecute el modelo de detección de objetos. Para este componente se deben considerar las restricciones de diseño del robot, ya que un exceso de peso o una mala distribución de los componentes elegidos puede ocasionar que la estructura se sumerja o que su funcionamiento y navegación se alteren. También es necesario considerar la velocidad de procesamiento de la computadora a elegir, ya que se debe cumplir con el requerimiento de identificar objetos en tiempo real (o lo más cercano al tiempo real), por lo que una respuesta tardía podría afectar al desempeño del sistema de visión.

A continuación se presenta un par de alternativas de computadoras que fueron consideradas además de la computadora electa:

- NVIDIA Jetson Nano: Es una microcomputadora desarrollada para aplicaciones embebidas que requieren un poder de procesamiento elevado en componentes pequeños. Permite correr múltiples modelos de aprendizaje de máquina y acelerar los cálculos requeridos por dichos modelos aprovechando el alto procesamiento que provee. Sus aplicaciones comunes son clasificación de imágenes, detección de objetos y procesamiento de lenguaje natural [16].

  La principal desventaja de este componente es un precio más elevado sobre la opción elegida, por lo que no se ajusta con la restricción del presupuesto existente. Otra desventaja es que se requiere de un largo tiempo de espera para tenerla por ser un producto de importación (aproximadamente 32 semanas de espera).

- ASUS Tinker Board S: Es una microcomputadora que cuenta con un procesador *quad-core* y *GPU* integrada, lo que le permite ser usada en aplicaciones de *gaming*, visión por computadora y procesamiento de imágenes gracias a su composición en hardware para aplicaciones dedicadas [4]. Una de sus desventajas es que también es un producto con costo elevado, por lo que no se ajusta a la restricción de presupuesto limitado. Otra desventaja es su compatibilidad para desarrollar aplicaciones sobre ella, ya que, al no ser un producto tan conocido, no posee tantos ejemplos de desarrollos de los cuales pudiera aprenderse o pudieran tomarse como base, lo que retrasaría el desarrollo del proyecto al requerir más tiempo para la documentación y realización de pruebas.

La microcomputadora elegida es la **Raspberry Pi 4 Model B** (mostrada en la Figura 4.4), la cual es una computadora de placa única (*SBC* por sus siglas en inglés) creada por la *Raspberry Pi Foundation* con el objetivo de que las personas tengan acceso a la tecnología de forma sencilla, además de que puedan desarrollar soluciones de acuerdo a sus necesidades [52]. Cuenta con un diseño compacto, poder de procesamiento elevado y compatibilidad con una gran cantidad de aplicaciones debido a su composición interna y su optimización

entre hardware y software. Provee una interfaz gráfica que favorece el desarrollo de aplicaciones y permite aprovechar el hardware, además de requerir un bajo consumo de energía y contar con un gran repositorio de proyectos desarrollados por la comunidad, lo que facilita la adopción de soluciones para las necesidades de las personas [44].

Figura 4.4: Microcomputadora Raspberry Pi 4 Model B a usar.

El último bloque a considerar para el móodulo de visión es la *Detección y clasificación de los objetos*, perteneciente al submódulo **sistema de detección**. Para este bloque se debe desarrollar un algoritmo rápido y eficiente que provea una alta precisión en el reconocimiento y clasificación de objetos, además de poder ser ejecutado en la microcomputadora. Para cumplir con los objetivos de la solución propuesta, la detección de los objetos debe seguir un proceso general que se describe a continuación:

1. El robot navega sobre superficies acuáticas en busca de basura y desechos.
2. La cámara digital proporciona imágenes de manera constante a la computadora, la cual tiene corriendo el algoritmo de detección de objetos.
3. La computadora provee la imagen recibida al modelo de detección de objetos con la intención de analizar la imagen e identificar todos los objetos de interés presentes en ella.
4. En caso de detectarse algún objeto, se clasifica de acuerdo con los tipos de basura y desechos considerados en este proyecto.

5. El resultado del objeto detectado se debe proporcionar al usuario que monitoree el funcionamiento del robot, lo que permite evaluar y comprobar el desempeño de modelo de detección de objetos. Si no se detecta algún objeto, el usuario también lo visualiza para corroborar que efectivamente no haya objetos de interés dentro de la imagen, así como identificar si el modelo requiere mejoras.

El proceso anterior debe ser un proceso cíclico, por lo que debe ser ejecutado en todo momento hasta recibir una interrupción del mismo. La razón de ser un proceso cíclico recae en el requerimiento de poder identificar objetos en tiempo real, lo que implica que constantemente se capturen imágenes del ambiente navegado y sean analizadas para tratar de identificar objetos de interés.

Teniendo en cuenta lo anterior, el modelo a trabajar está basado en el uso de redes neuronales convolucionales, además de usar arquitecturas predefinidas para tareas de detección y clasificación de objetos. Específicamente en este proyecto se usan diversas arquitecturas de YOLO [54] [20] [8], las cuales son implementaciones de redes neuronales optimizadas para identificar y clasificar objetos de forma rápida y con buena precisión según sea la arquitectura.

El nombre de YOLO proviene de la frase *You only look once* debido a su capacidad de identificar objetos sin la necesidad de realizar un procesamiento complejo de las imágenes. Usar un sistema de detección de objetos con una arquitectura definida como YOLO tiene ciertas ventajas, donde la más importante es porder generar buenos resultados gracias a sus implementaciones previas y a las mejoras que han recibido las arquitecturas. De esta manera, se puede reducir el tiempo que tomaría crear un modelo de detección de objetos desde el inicio, permitiendo destinar más tiempo al diseño de la base de datos, al entrenamiento del sistema y a la realización de pruebas. La Figura 4.5 muestra la arquitectura de red neuronal de YOLO en su primera versión, la cuál consta de una serie de operaciones de convolución en cada capa de la red, así como operaciones de subsampling para reducir el número de mapas de características, con la intención de generar una identificación y clasificación del objeto rápidamente.

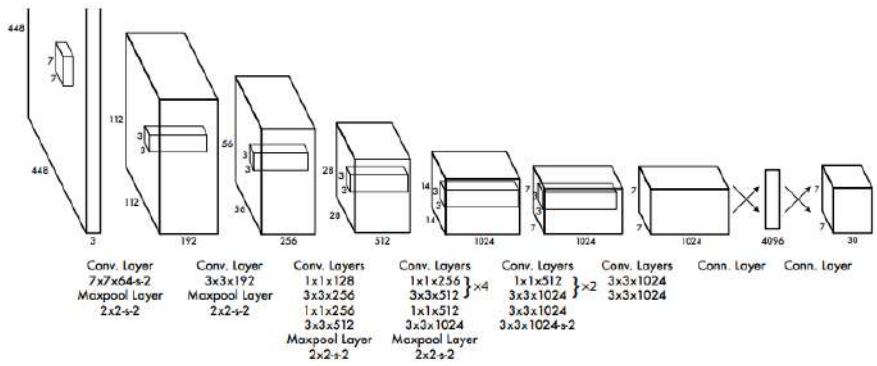


Figura 4.5: Arquitectura de red neuronal de YOLO v1 [54].

Además de las características anteriores, YOLO es un sistema de detección ligero, por lo que no requiere alto poder de procesamiento en hardware para ser ejecutado. Esta característica es fundamental ya que se contempla el uso de una microcomputadora con poder de procesamiento limitado, pero que cuenta con la potencia suficiente para ejecutar el modelo de clasificación de objetos de forma constante. Una ventaja adicional de usar YOLO es que existe una gran variedad de implementaciones previas en proyectos de detección de objetos, desde clasificación de automóviles en zonas de tráfico [67] hasta la identificación de rostros para composición de carteles de películas [51], lo que permite tomar desarrollos previos para facilitar su implementación y obtener mejores resultados.

Para este proyecto se tomó la decisión de trabajar con las arquitecturas de YOLOv3 [20] y YOLOv4 [8], las cuales son el resultado de mejorar la velocidad, la precisión y el desempeño de la primera versión. La intención de trabajar con ambas versiones es comparar su funcionamiento a la hora de reconocer objetos en tiempo real, además de evaluar cuál de las dos arquitecturas se ajusta mejor a las necesidades del problema.

4.4. Estructura del robot

Continuando con el diagrama general de la solución, el siguiente bloque a explicar corresponde al bloque de la *Estructura del robot*, el cual tiene el propósito de representar el modelo físico del sistema robótico sobre el que se montarán

los componentes electrónicos y los elementos periféricos. También es necesario considerar que el robot debe cumplir con los requerimientos funcionales del proyecto, en especial con la necesidad de navegar sobre superficies acuáticas sin problema, por lo que el diseño debe considerar una buena distribución de los componentes y sus pesos, además de un así como un correcto sistema de flotación e inhibición de filtraciones de agua.

El diseño de la estructura del robot para este proyecto está basado en un par de robots propuestos para navegación de superficies acuáticas, lo que permite adaptar los diseños de ambos y generar un modelo que cumpla con los requerimientos de la solución propuesta. El primero de ellos es el modelo desarrollado por Maindalkar y Ansari [40], el cual consiste en un **robot pescado** para monitoreo y evaluación de contaminación de ambientes acuáticos, y cuyo diseño puede ser observado en la Figura 4.6.

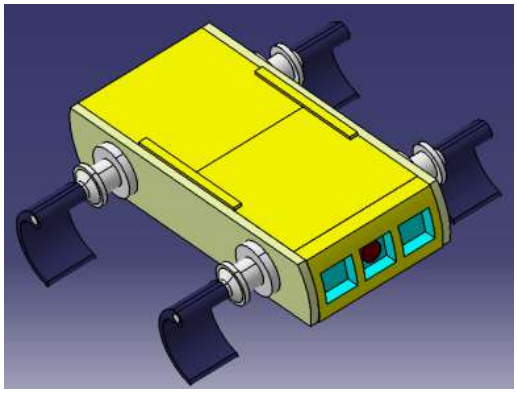


Figura 4.6: Diseño 3D del robot pescado [40].

El modelo del robot pescado destaca por tener un diseño fácil de replicar y su bajo costo en componentes y materiales usados. Involucra el uso de una cámara CMOS, un microcontrolador Arduino UNO en conjunto con sensores y componentes periféricos, y una microcomputadora Raspberry Pi. Cuenta con un diseño rectangular impulsado por cuatro aletas motorizadas, además de una estructura transparente hecha con polimetilmetacrilato que facilita la visión de la cámara usada. Como se mencionó anteriormente, la estructura de este robot en conjunto con los componentes usados es una buena propuesta de diseño, por lo que adaptarlo a las necesidades de este proyecto resultaría sencillo y permitiría

cumplir con algunos de los objetivos propuestos.

El segundo modelo de referencia es el propuesto por Himanshu Burde [11], el cual consiste en un **robot anfibio** para entregas en ambientes acuáticos, y cuyo diseño puede ser observado en la Figura 4.7. Este modelo se caracteriza por proponer un diseño de robot híbrido, lo que le permite navegar en superficies acuáticas con ayuda de una balsa externa propulsada con una hélice motorizada, además de moverse sobre ambientes terrestres con ayuda de llantas desplegables. Lamentablemente este modelo no especifica los componentes ni los materiales para la estructura, sin embargo se puede tomar su diseño como base y adaptarlo a las necesidades y objetivos planteados.

Figura 4.7: Diseño 3D del robot anfibio [11].

El propósito de tomar estos modelos como base es agilizar el proceso de diseño de un robot desde el inicio, no obstante la meta final es adaptar, modificar y mejorar dichos modelos para generar un prototipo que cumpla los requerimientos de este proyecto. Las mejoras y adaptaciones que requieren los modelos presentados anteriormente se enlistan a continuación:

- Usar materiales para la estructura del robot que sean resistentes al agua, que sean transparentes, y que permitan la captura de fotografías sin mucha distorsión de los objetos.
- Desarrollar un sistema de flotación auxiliar que ayude al robot a navegar sobre superficies sin hundirse.

- Mejorar la posición de los componentes electrónicos con la intención de distribuir el peso total del robot y permitir su navegación libremente.
- Agregar un sistema de iluminación compuesto por sensores y LEDs que proporcionen mayor iluminación al robot según la iluminación ambiental.
- Usar un único motor para realizar la propulsión del robot y reducir costos.

Para cumplir con las mejoras enlistadas, la estructura del robot consta de placas de polimetilmetacrilato transparente con un espesor de 3mm, las cuales están unidas con pegamento y plastilina epóxicos para evitar la filtración de agua y asegurar una buena fijación entre ellas. El sistema de flotación está compuesto por barras de nylamid fijas a las placas de polimetilmetacrilato, además de tubos de polietileno adheridos a dichas barras, permitiendo que el robot permanezca sobre la superficie acuática sin hundirse.

La distribución de los componentes electrónicos debe realizarse considerando el diseño final del robot y el espacio disponible dentro de el, además de tomar en cuenta el peso total de todos los componentes y el peso de la estructura misma. De esta manera, se puede asegurar una distribución inteligente para evitar que el robot se incline o no pueda moverse de forma correcta. Por último, la ubicación de los sensores de iluminación y los LEDs debe ser similar a la ubicación de la cámara usada, lo que proporcionará mayor visibilidad para detectar los objetos.

4.5. Movimiento y navegación del sistema

El siguiente módulo a explicar corresponde al *Movimiento y navegación del sistema*, compuesto por tres submódulos encargados de ofrecer movimiento al robot, generar información de la iluminación del ambiente, y administrar las conexiones electrónicas para energizar cada elemento del sistema. De acuerdo con el diagrama general de la solución, los bloques asociados a este módulo son los bloques con fondo azul, por lo que a continuación se incluye su explicación con la intención de justificar los elementos que componen cada subsistema.

El primer bloque de este módulo está asociado con los *Actuadores* del sistema, los cuales son los encargados de ofrecer movimiento y velocidad al robot. Considerando que el diseño del robot tiene contemplado usar un único motor

para llevar a cabo el movimiento, el componente a usar debe ofrecer la suficiente velocidad y fuerza para mover toda la estructura. De esta manera, el actuador elegido fue el **Motorreductor RS-45PA-18140**, el cual puede observarse en la Figura 4.8.

Figura 4.8: Motorreductor RS-45PA-18140.

El segundo bloque de este módulo es el bloque de los *Sensores e iluminación*, el cual es el encargado de ofrecer información sobre la luz ambiental y evaluar si es necesario aumentar la iluminación externa para mejorar la calidad de las imágenes capturadas por la cámara. Para este bloque se tiene contemplado el uso de dos **Sensores de luz OKY-3101**, caracterizados por su bajo costo de venta y altos niveles de sensibilidad y confiabilidad en las mediciones. La Figura 4.9 muestra uno de los sensores de iluminación a usar.

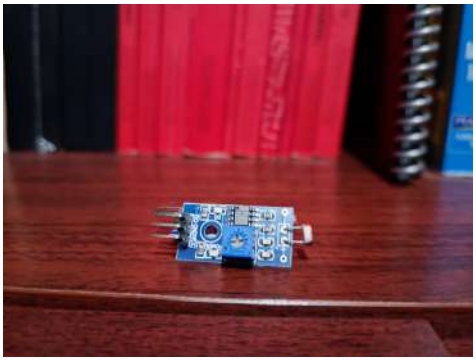


Figura 4.9: Sensor de luz OKY-3101.

El último bloque del módulo corresponde al *Microcontrolador*, el cual tiene el objetivo de integrar las funcionalidades de movimiento del actuador y recepción de información de los sensores, permitiendo la toma de decisiones para incrementar o decrementar la velocidad de navegación y la luz exterior proporcionada. De acuerdo con las restricciones del proyecto, el microcontrolador a usar es un **Atmega328** integrado a la placa **Arduino UNO**, el cual facilita el desarrollo de este proyecto gracias a su compatibilidad con componentes electrónicos y su interfaz de programación. La Figura 4.10 muestra la placa Arduino UNO que se usará en este proyecto.

Figura 4.10: Arduino UNO con el microcontrolador Atmega328.

Es importante mencionar que todos los componentes de este módulo requieren ser energizados con una fuente de poder auxiliar qe sea lo suficientemente potente para energizar cada uno ellos y proveer el voltaje necesario para evitar daños en la electrónica. Tomando en cuenta el presupuesto limitado, así como los requerimientos de los componentes elegidos, se tomó la decisión de trabajar con una **Batería Harden de 6.7 volts**, la cual es una batería recargable de ácido y plomo que se caracteriza por su bajo costo, peso reducido y potencia suficiente. La Figura 4.11 muestra la batería que se usará en la implementación de este proyecto.

Figura 4.11: Batería Harden de 6.7 volts.

4.6. Módulo de comunicación

El último módulo a abordar corresponde a la *Comunicación* del sistema, el cual tiene la intención de definir la forma de transmisión de la información entre los distintos componentes elegidos, así como establecer el flujo que debe seguir dicha información para que pueda ser usada de forma adecuada y se cumpla con los objetivos planteados. De acuerdo con la información presentada en la subsección *Descripción del diseño de la solución* de este apartado, el módulo de comunicación tiene a su cargo dos funciones principales: transmitir las imágenes capturadas con la cámara hacia la unidad de procesamiento, y mostrar los resultados de la clasificación de basura en una pantalla externa.

Para transmitir las imágenes capturadas por la cámara se tienen contempladas dos formas de comunicación. La primera de ellas es una comunicación alámbrica entre la cámara elegida y la microcomputadora, permitiendo una alta velocidad de transmisión de la información y un menor consumo de energía de la cámara. La segunda es una comunicación inalámbrica mediante Wi-Fi que permita eliminar el cableado y reducir el riesgo de un posible cortocircuito por el contacto de los cables con el agua. Ambos tipos de comunicación pueden ser implementados gracias a las características técnicas de las cámaras elegidas, además del módulo Wi-Fi y de los puertos USB integrados a la microcomputadora, lo que facilita ambos tipos de comunicación.

Por otro lado, para la transmisión de resultados del modelo de detección de objetos se consideró una conexión alámbrica entre la microcomputadora y una pantalla o monitor externos, facilitando la visualización de resultados al usuario y permitiendo evaluar el desempeño del modelo. Para este proceso se tiene considerado aprovechar el puerto HDMI con el que cuenta la microcomputadora elegida, de tal manera que cada resultado obtenido a la salida del modelo pueda ser visualizado instantáneamente.

4.7. Estándares utilizados

La realización de este proyecto involucra el uso de recursos estandarizados que faciliten la reproducción parcial o total del sistema, además de proveer un fácil acceso a las herramientas elegidas y que sea compatible con la mayoría de las soluciones ya realizadas. Los recursos estandarizados que se presentan a continuación se dividieron en dos grupos: los recursos estandarizados por *organismos internacionales*, que son aquellos que cuentan con publicaciones o avalaciones internaciones, y los recursos estandarizados *De Facto* o *estándares abiertos*, que son los recursos usados por una gran cantidad de personas pero que no cuentan con una avalación internacional, es decir que son de libre acceso para su modificación y mejora.

4.7.1. Estándares internacionales

- **ISO/IEC 14882:2017:** Conocido como ISOC++, es el estándar propuesto por *ISO* para describir los requisitos, desarrollo e implementación del lenguaje de programación C++ [31]. Se contempla su uso en el desarrollo de código para el microcontrolador con la intención de controlar los movimientos del robot y obtener información de los sensores elegidos.
- **IEEE 802.11n:** Conocido como Wi-Fi, es un estándar desarrollado por *IEEE* que ofrece una comunicación inalámbrica para la transmisión de información. El estándar propone una velocidad de transmisión máxima de 600 Mbps, y alcanza una velocidad estable entre 80 y 100 Mbps. Otra caracterísitica de este estándar es que, además de trabajar en la

banda de frecuencias de los 2.4 GHz, permite trabajar en la banda de frecuencias de los 5 GHz, asegurando la transmisión de información en una banda menos saturada como occurre con la banda de frecuencias de los 2.4 GHz [72].

Este estándar se usará para la transmisión de imágenes entre la microcomputadora y la cámara fotográfica. Es importante mencionar que la comunicación inalámbrica puede tener repercusiones en el desempeño del modelo, por lo que es necesario realizar una serie de pruebas para determinar la mejor opción para implementar.

- **JPEG ISO/IEC 10918-1:** Es un estándar de compresión de información recomendado por ITU y desarrollado en conjunto con ISO y JPEG, el cual permite comprimir imágenes mediante un algoritmo de compresión por pérdida y reducir el tamaño de las mismas, además de proporcionar un modelo de representación de la información que pueda ser usado por cámaras fotográficas digitales y dispositivos de captura de imágenes [32]. Las imágenes obtenidas con la cámaras fotográficas elegidas se generan digitalmente con la extensión *.JPG, facilitando su procesamiento gracias a la gran variedad y compatibilidad de herramientas desarrolladas para trabajar con este tipo de imágenes.

- **HDMI (EIA/CEA-861)**: Conocido como HDMI, es un estándar desarrollado inicialmente por las empresas *Hitachi*, *Matsushita*, *Philips*, entre otras, y actualmente licenciado por la organización *HDMI.org*. Tiene el objetivo de desarrollar una interfaz de audio y video para transmitir información digital sin compresión con alta calidad. También permite estandarizar el uso de una misma conexión y reemplazar a los estándares de video analógico [28]. HDMI implementa los estándares EIA/CEA-861 de la *Video Electronics Standards Association*, los cuales son estándares que definen los formatos para video y formas de onda para transmisión de información, así como definir la forma de transportar audio y video comprimidos [28]. Las versiones a usar de este estandar son las versiones *HDMI tipo A* en la pantalla externa y *HDMI tipo D* en la microcomputadora.

- **USB 2.x/3.x**: Es un estándar desarrollado con la intención de homogeneizar las conexiones entre dispositivos electrónicos, reemplazando puertos complejos de pines por puertos únicos y compatibles [14]. El estándar define las especificaciones para cables y conectores, además de establecer protocolos de comunicación y transporte de energía entre dispositivos [22]. En este proyecto las versiones a usar de USB son las versiones 2.x y 3.x, las cuáles se diferencían por su velocidad de transmisión de datos y su método de comunicación y serialización.

4.7.2. Estándares *De Facto* o Abiertos

- **Python**: Creado por Guido van Rossum, es un lenguaje de programación de alto nivel que se caracteriza por su administración de memoria automática y soportar múltiples paradigmas de programación [24]. Posee una licencia de código abierto proporcionada por *Python Software Foundation*, lo que permite que usuarios de todo el mundo puedan realizar mejoras o extensiones al código proporcionado con propósitos específicos [49]. En este proyecto se usará la distribución Python 3.x para el uso de herramientas de etiquetado de datos y tratamiento de las imágenes.

- **Ubuntu:** Es un sistema operativo creado por Mark Shuttleworth bajo la organización Canonical con el objetivo de fomentar el uso de software libre y crear herramientas que fueran de fácil acceso para la gente. Al ser un sistema base Linux, permite la compatibilidad con múltiples herramientas desarrolladas por la comunidad, además de permitir un ambiente amigable para el desarrollo de aplicaciones [69]. Para este proyecto se usará la distribución de Ubuntu 20, la cual es una distribución que facilita el despliegue de aplicaciones y es compatible con las computadoras Raspberry Pi.

Capítulo 5

RECOLECCIÓN Y PREPARACIÓN DE LAS IMÁGENES

De acuerdo con la metodología en cascada mencionada en el apartado de *Introducción*, después de recabar los requerimientos del proyecto, establecer una planeación de trabajo, y modelar el diseño de la solución propuesta, la siguiente etapa es la construcción de la solución. El primer paso en esta etapa es el desarrollo del modelo de reconocimiento y clasificación de objetos, el cual está asociado al módulo de visión presentado en el capítulo anterior. Para generar dicho modelo, se consideró seguir una metodología que establezca las bases para un correcto desarrollo de modelos de aprendizaje de máquina, optando por seguir el **Proceso de ciencia de datos** propuesto por Vijay Kotu [36].

El Proceso de Vijay Kotu consiste en una serie de actividades iterativas con el objetivo de generar modelos de inteligencia artificial que nos ayuden a encontrar patrones y relaciones en la información objetivo. La elección de este proceso se debe a que se ajusta adecuadamente con la metodología en cascada seguida a lo largo de todo el proyecto, además de que el proceso está compuesto por una serie de actividades generales y de fácil replicabilidad, lo que permite

que se adecúe de forma sencilla al desarrollo de este proyecto. Los elementos que involucra el proceso de ciencia de datos mencionado anteriormente se enlistan a continuación:

1. Entender el problema al cual se dará solución.
2. Preparar muestras de los datos que sirvan como ejemplos para entender el entorno.
3. Desarrollar un modelo de aprendizaje de máquina que sea capaz de resolver, ya sea parcial o totalmente, el problema encontrado.
4. Aplicar el modelo desarrollado sobre la colección total de datos y, de ser posible, aplicar en el mundo real.
5. Desplegar y dar mantenimiento al modelo desarrollado para mejorar su desempeño.

Como puede notarse, el primer elemento de la lista fue abordado en el apartado *Análisis del problema*, donde se establecieron los requerimientos de la solución, se mencionaron las restricciones existentes para su desarrollo y se expusieron proyectos similares a la solución propuesta. En consecuencia, la siguiente tarea consiste en generar una muestra de imágenes que puedan ser usadas para el entrenamiento del modelo de detección de objetos. El presente capítulo tiene la intención de documentar las bases de datos de imágenes de basura que fueron encontradas, además de proporcionar una explicación de las técnicas usadas para preparar las imágenes elegidas, y mostrar ejemplos antes y después de aplicar dichas ténicas.

5.1. Recolección de imágenes

Uno de los retos en visión por computadora es tener conjuntos de imágenes (también llamados *Datasets*) compuestos por una gran variedad de ejemplos, lo que permite desarrollar modelos de aprendizaje de máquina con alta precisión y poco susceptibles a errores. El reto aumenta cuando el área de interés no ha sido lo suficientemente investigada como lo han sido otras áreas, causando que

la cantidad de información generada por proyectos previamente desarrollados sea menor. Bajo esta línea, y considerando los capítulos anteriores, puede concluirse que la investigación y desarrollo de soluciones que involucren detección de basura en ambientes acuáticos con poca iluminación es un área que no ha sido lo suficientemente abordada, por lo que la información y recursos con los que se cuenta para elaborar un modelo de visión robusto son limitados.

Buscando reducir el tiempo que tomaría generar un dataset desde el inicio, se optó por buscar en Internet datasets disponibles de imágenes de basura y desechos. El primer requerimiento que debían cumplir los datasets es que pudieran ser usados en investigaciones científicas y académicas. Otro requerimiento es que el ambiente en el que fueron capturados los objetos fuera un ambiente con poca luz, o en su defecto que hayan sido capturados con un fondo oscuro que simule el ambiente con ausencia de luz. Un requerimiento adicional a cumplir fue que los datasets debieran estar compuestos por una gran variedad de objetos, los cuales sirvieran como ejemplos para mejorar la precisión del modelo.

Después de una búsqueda exhaustiva, se puede asegurar que los datasets compuestos por imágenes de basura y desechos son escasos, por lo que obtener un dataset con los requerimientos previamente descritos es una tarea con mayor dificultad. Se encontraron aproximadamente 20 datasets de imágenes de basura, los cuales fueron generados por toma de fotografías propias o por recolección de imágenes en Internet. Debido a que la mayoría de los datasets encontrados fueron generados mediante la recolección de imágenes, la información que contienen es bastante similar, causando redundancia en las imágenes y reduciendo la precisión de un posible modelo usando dichos conjuntos de imágenes.

De los 20 datasets encontrados, los 3 datasets que se consideró que aportaban mayor variedad en los objetos fueron *TACO* [48], *Garbage classification* [33] y *WaDaBa* [7]. Una característica en común de estos datasets es que fueron desarrollados con el propósito de detección y clasificación de objetos, por lo que su uso en este proyecto era viable. Lamentablemente, de los 3 datasets, el único que cumplía con los requerimientos mencionados previamente fue el dataset **WaDaBa**, por lo que todas las actividades que implicó el desarrollo del modelo de detección y clasificación de objetos fueron usando este dataset.

5.2. Descripción de la base de datos

WaDaBa es una base de datos de imágenes de basura y desechos plásticos generada mediante la toma de fotografías, la cual está compuesta por 4000 imágenes distribuidas uniformemente entre 100 objetos distintos. De cada objeto se capturaron 40 fotografías con un fondo oscuro, variando la posición del objeto dentro de la imagen, modificando la iluminación ambiental, deformando el objeto, y capturando las imágenes con mayor o menor acercamiento de la cámara. Ejemplos de las imágenes que conforman la base de datos WaDaBa pueden observarse en la Figura 5.1.

Figura 5.1: Ejemplos de imágenes de la base de datos WaDaBa [7].

La clasificación de los objetos que sigue WaDaBa es mediante los tipos de plástico que existen, los cuales son: 1) *PET*; 2) *PE-HD*; 3) *PVC*; 4) *PE-LD*; 5) *PP*; 6) *PS* y; 7) *O*. Una ventaja de usar este dataset es que el tamaño de archivo de cada imagen es relativamente pequeño (aproximadamente 1 MB por imagen), por lo que el tamaño total en memoria ocupado por el dataset completo es menor en comparación con el tamaño ocupado por otros datasets. En cambio, una desventaja del dataset es que la cantidad de imágenes por tipo de objetos (de ahora en adelante llamados *clases*) está desbalanceada, lo que implica que existe una mayor cantidad de imágenes de una misma clase de plásticos en comparación con la cantidad de imágenes de otras clases.

Específicamente, la clase que cuenta con más imágenes en el dataset es la clase PET (57 objetos distintos), mientras que las clases PVC y PE-LD no cuentan con objetos capturados dentro del dataset. El resto de los objetos cap-

turados corresponden a las clases PE-HD (15 objetos distintos), PP (14 objetos distintos) y PS (13 objetos distintos). El dataset incluye una clase extra considerada como *OTROS*, sin embargo solo está compuesta por 40 fotografías de un objeto no plástico que no aporta suficiente información. La Figura 5.2 muestra una gráfica de barras que representa la cantidad de objetos distintos que fueron fotografiados y forman parte del dataset WaDaBa.

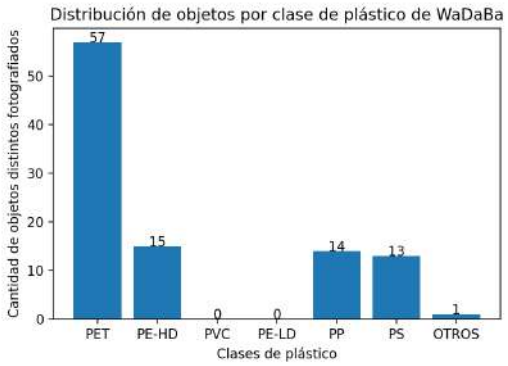


Figura 5.2: Distribución de objetos por clase de plástico de WaDaBa.

De la gráfica anterior se puede observar que la mayor cantidad de objetos están concentrados en la clase PET, contando con más del triple de información de la que cuentan otras clases. Usar esta distribución de datos para desarrollar un modelo de detección resultaría en un sesgo importante a la hora de clasificar objetos, provocando que la mayoría sean clasificados como clase PET, o incluso inhibiendo la presencia de las otras clases sin importar el tipo de objeto que se encuentre en la imagen.

5.3. Preparación de las imágenes a usar

Con la intención de reducir la complejidad del problema y disminuir la variedad de objetos a detectar, se limitó la cantidad de clases de plástico a ser reconocidas por el modelo de detección de objetos. Las clases que se usaron durante el desarrollo de este proyecto fueron PET, PS y PE-HD, las cuales se eligieron considerando que los objetos de la misma clase poseen características

similares que los hacen fáciles de identificar, pero son lo suficientemente complejos como para ser clasificados erróneamente según sea su deformación y la posición en la que se encuentren a la hora de ser fotografiados.

5.3.1. Selección de las imágenes

Después de un análsis exploratorio de las imágenes, se encontró que algunos objetos poseían combinaciones de colores y formas bastante similares entre ellos, lo que podría traducirse como información repetida a la hora de entrenar el modelo de detección de objetos. Para evitar este problema, se seleccionaron aquellos objetos que proporcionaran mayor variabilidad de información, ya sea por su forma, su tamaño, su tipo de deformación, el ángulo de la fotografía, o por composición de colores.

Como resultado de la selección estratégica de las imágenes, la cantidad de objetos fue reducida considerablemente para la clase PET, eligiendo únicamente 28 de los 57 objetos distintos que conformaban la clase. Por otro lado, algunos objetos para las clases PS y PE-HD también fueron descartados, sin embargo la cantidad de objetos para ambas clases no se vio tan afectado. Para la clase PS se eligieron 12 de los 13 objetos que componían la clase, mientras que para la clase PE-HD se eligieron 11 de los 15 objetos que componían la clase. Las Figuras 5.3 y 5.4 muestran un par de gráficas de barras que representan respectivamente la cantidad de objetos y de imágenes, divididas por clase de plástico, que se usaron en este proyecto.

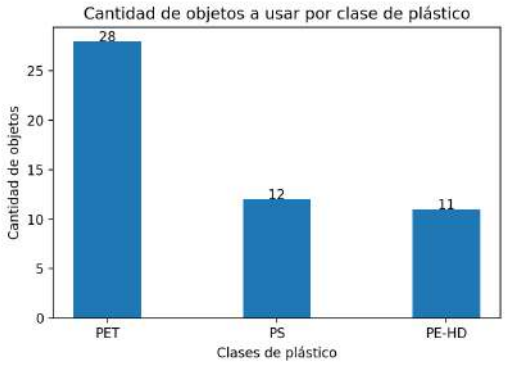


Figura 5.3: Distribución de objetos a usar por clase de plástico.

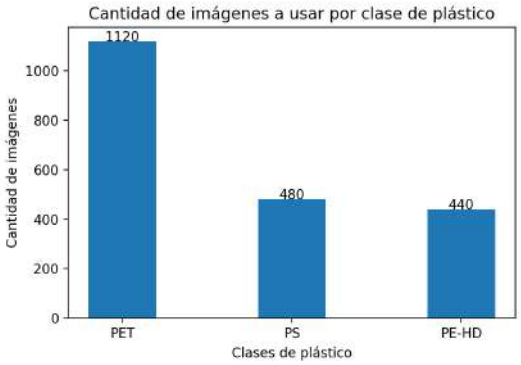


Figura 5.4: Distribución de imágenes a usar por clase de plástico.

De las gráficas anteriores puede notarse que las distribuciones de objetos y de imágenes entre las clases de basura continuan desbalanceadas, sin embargo, después de la selección estratégica de imágenes, la proporción de objetos a usar para la clase PET con respecto a las otras clases disminuyó considerablemente. Algunas ventajas de usar esta distribución de imágenes para entrenar un modelo son: reducción del sesgo hacia la clase PET en la clasificación de objetos; baja complejidad para la generalización de los objetos a detectar; y fácil identificación de necesidades o deficiencias del modelo, ya sea por cantidad de información o por balanceo de imágenes en cada clase.

5.3.2. Tratamiento de las imágenes

Una vez que fueron seleccionadas las imágenes a usar para el entrenamiento del modelo, la siguiente tarea fue realizar un reescalamiento de cada una de ellas. El reescalamiento de las imágenes ayuda a reducir su complejidad de procesamiento, así como estandarizar el tamaño de cada imagen y disminuir el espacio en memoria que ocupan, permitiendo que sea más rápido y sencillo trabajar con ellas y manipularlas.

Cada imagen del dataset de WaDaBa tiene un tamaño de 1920 pixeles de largo y 1277 pixeles de alto, por lo que trabajar con este tamaño de imágenes resultaría en un alto costo computacional y requeriría más tiempo por cada acción a realizar sobre las imágenes. Buscando evitar estos inconvenientes, se tomó la decisión de reescalar las imágenes a un tercio de su tamaño original,

es decir a 640 pixeles de largo y 426 pixeles de alto. El reescalamiento de las imágenes se hizo con ayuda de un código en Python que puede consultarse en el Apéndice A bajo la etiqueta A.1. Un ejemplo de las imágenes a usar en este proyecto pueden observarse en las Figuras 5.5, 5.6 y 5.7, las cuales son fotografías de objetos de las clases PET, PS y PEHD respectivamente.

5.3.3. Etiquetado de las imágenes

La última actividad necesaria para preparar las imágenes fue el etiquetado de cada una de ellas, siguiendo un formato que fuera compatible con la arquitectura a usar. De acuerdo con lo expuesto en el apartado *Diseño general de la solución*, la arquitectura de redes neuronales a implementar para el módulo de visión está basada en las arquitecturas YOLO, por lo que el etiquetado de las imágenes debe atender al formato requerido por esta arquitectura para facilitar la identificación de los objetos.

Siguiendo la definición de YOLO, la detección de los objetos se hace mediante *Bounding Boxes* [54], los cuales pueden entenderse como recuadros o ventanas encargadas de definir el espacio que ocupa un objeto dentro de toda la imagen. La intención de usar bounding boxes es señalar la presencia de un objeto basado en sus coordenadas y tamaño dentro de la imagen, por lo que es necesario indicar al modelo la información correspondiente de todos los objetos para que puedan ser identificados y clasificados de forma correcta.

El formato de etiquetado requerido para usar las arquitecturas de YOLO sigue la siguiente estructura:

$$<class\_number> <x\_coord> <y\_coord> <width> <height>,$$

donde $<class\_number>$ se refiere al número que identifica la clase a la que se está asociando el objeto marcado, $<x\_coord>$ y $<y\_coord>$ se refieren a las coordenadas de la esquina superior izquierda del bounding box que encierra al objeto, y $<width>$ y $<height>$ se refieren a la base y altura del bounding box que encierra al objeto.

Figura 5.5: Ejemplo de imagen a usar de la clase PET.

Figura 5.6: Ejemplo de imagen a usar de la clase PS.

Figura 5.7: Ejemplo de imagen a usar de la clase PEHD.

Con la intención de etiquetar los objetos dentro de las imágenes de forma rápida y precisa, se usó la herramienta *LabelImg* [68], la cual es una herramienta diseñada con la intención de agilizar el proceso de etiquetado de imágenes usando bounding boxes. Una de las características principales de LabelImg es que tiene compatibilidad con varios formatos de etiquetado, dentro de los cuales se encuentra el formato requerido por las arquitecturas de YOLO, por lo que es una herramienta ideal para agilizar el proceso de etiquetado de cada uno de los objetos en las imágenes.

Es importante destacar que, por cada imagen usada, se debe generar un archivo de texto que incluya la información de todos los objetos etiquetados dentro de ella. En caso de etiquetar más de un objeto en la misma imagen, la información de cada uno de ellos se inlcuye en el mismo archivo de texto separada por un salto de línea, lo que permite indicar al modelo la presencia de múltiples objetos que deben ser idenficiados de forma independiente. Un ejemplo del etiquetado de los objetos a detectar se puede observar en la Figura 5.8, la cual muestra una botella de PET etiquetada en el formato requerido por YOLO usando la herramienta LabelImg.

Figura 5.8: Ejemplo de etiquetado de objeto de clase PET usando LabelImg.

La información incluida en el archivo de texto correspondiente a la imagen mostrada en la Figura 5.8 es la siguiente:

$$0 \quad 0.507812 \quad 0.556338 \quad 0.509375 \quad 0.530516,$$

donde el primer valor hace referencia a la clase PET, la cual está asociada al número 0, mientras que la clase PS está asociada al número 1 y la clase PEHD está asociada al número 2. Los dos valores después de la clase hacen referencia a las coordenadas de la esquina superior izquierda del bounding box que encierra al objeto, los cuales son valores normalizados respecto al tamaño total de la imagen. Los últimos dos valores corresponden a la base y altura del bounding box que encierra al objeto, los cuales también son valores normalizados respecto al tamaño total de la imagen.

Al finalizar el etiquetado de las 2040 imágenes seleccionadas, se generaron respectivamente 2040 archivos de texto distintos que incluyen la información de los objetos etiquetados en cada imagen. Las imágenes del dataset WaDaBa contienen solamente un objeto, por lo que cada archivo de texto posee únicamente la información del objeto etiquetado dentro de la imagen.

5.3.4. Imágenes complementarias

Con la intención de mejorar la precisión del modelo de detección de objetos, se tomó la decisión de complementar el conjunto de imágenes de basura con imágenes adicionales, las cuales debieran contener objetos que proporcionen mayor variabilidad a la hora de entrenar el modelo. La intención de agregar estas imágenes complementarias es ayudar al modelo a entender el contexto en el cual los objetos deben ser identificados, además de proporcionar mayor información sobre los objetos que no deben ser clasificados de acuerdo con alguna de las clases objetivo.

Para llevar a cabo esta tarea, se recolectaron 2040 imágenes con distintas características como lo son el tamaño, la cantidad de objetos, la paleta de colores, e incluso la orientación de la imagen. Todas las imágenes que fueron seleccionadas cumplen con la condición de no poseer objetos de algún tipo de plástico a identifcar, además de estar compuestas por características visuales que ayudan al modelo a incrementar la precisión en la identificación de los objetos.

Debido a que las imágenes complementarias fueron recolectadas usando un motor de búsqueda, cada imagen posee tamaños y orientaciones distintas. Para que pudieran ser añadidas al dataset, fue necesario estandarizarlas de acuerdo con los tamaños de las imágenes de objetos de basura a usar, además de generar el etiquetado correspondiente de cada una de ellas. Considerando que las imágenes adicionales son imágenes que no poseen objetos que nos interese detectar, el archivo de texto asociado a cada una de ellas debe ser un archivo de texto en blanco, lo que le indica al modelo que no hay objetos a detectar dentro de la imagen pero que se puede usar la información proporcionada para aumentar la precisión de la identificación de objetos.

Para realizar la estandarización de las imágenes adicionales se usó un código en Python, el cual consiste en evaluar el tamaño y orientación de cada una de ellas y reescalarlas considerando los tamaños objetivo (640 pixeles de largo y 426 pixeles de alto). El código también es el encargado de generar los archivos de texto en blanco, tomando el nombre de cada imagen y asociándolo respectivamente al archivo de texto correspondiente. El código usado puede consultarse en el Apéndice A bajo la etiqueta A.2. Ejemplos de las imágenes adicionales pueden observarse en las Figuras 5.9, 5.10 y 5.11, las cuales son imágenes compuestas por diversos objetos que fueron usadas para el entrenamiento del modelo de detección de objetos.

Figura 5.9: Ejemplo 1 de imagen adicional.

Figura 5.10: Ejemplo 2 de imagen adicional.

Figura 5.11: Ejemplo 3 de imagen adicional.

5.3.5. Generación artificial de imágenes

Como se ha visto a lo largo de este capítulo, un reto importante para el desarrollo de un modelo de detección de objetos es contar con suficientes imágenes que proporcionen variabilidad durante el entrenamiento, lo cual ayudaría a mejorar la precisión para la identificación y clasificación de los objetos. Si no se cuenta con suficiente información, pueden surgir varios problemas que no permitan al modelo generalizar la información proporcionada durante el entrenamiento.

Un problema típico a la hora de entrenar modelos para detección de objetos es el sobreentrenamiento u *overfitting*, el cual ocurre cuando el modelo no fue capaz de generalizar la información de las imágenes durante el entrenamiento y únicamente la memorizó. Cuando un modelo cae en el sobreentrenamiento, tiene un buen desempeño a la hora de detectar objetos sobre las imágenes usadas en

la fase de entrenamiento, pero tiene un mal desempeño usando imágenes de prueba o imágenes nunca antes usadas [61].

Con la intención de evitar el sobreentrenamiento y proporcionar más opciones para desarrollar diversos modelos, se optó por generar información artificial con base en la información previamente etiquetada. Este proceso es conocido como **generación de datos artificiales** o *data augmentation* en inglés, y tiene el propósito de generar más imágenes que puedan ser usadas para la fase de entrenamiento del modelo, con lo cual se proporciona mayor variabilidad entre las imágenes y se ayuda a mejorar la precisión del mismo [61].

La generación de datos artificiales es funcional cuando se cuenta con pocos datos como es el caso de este proyecto, donde la cantidad de imágenes disponibles para generar modelos de visión por computadora limitan el poder de generalización de la solución. Cabe destacar que la integración de imágenes artificiales en este proyecto también facilitar la aproximación de la solución al tratarse de un problema de detección de objetos en ambientes no idóneos o poco comúnes.

Para llevar a cabo un proceso de generación de datos artificiales, es necesario realizar varias acciones o técnicas sobre las imágenes originales que conforman el dataset, de tal manera que se generen nuevas imágenes sin provocar dependencia lineal entre la información a usar para el entrenamiento [61]. Algunas de las técnicas de procesamiento de imágenes más comunes para realizar data augmentation son: rotarlas o cambiar su orientación, modificar sus tamaños, agregar ruido, usar uno o varios filtros y modificar los canales de colores que las componen, entre otras.

Intentando facilitar el proceso de data augmentation, se tomó la decisión de usar *AugLy* [6], la cual es una herramienta desarrollada por ingenieras de *Facebook* con la intención de generar información adicional mediante operaciones a conjuntos de datos. AugLy cuenta con más de 30 operaciones distintas para imágenes que pueden usarse de forma individual o mezcladas, lo que permite generar nuevas imágenes que proporcionen más información al modelo pero que no sean interpretadas como información repetida.

El proceso de data augmentation fue aplicado para todas las imágenes complementarias, aplicando las técnicas de procesamiento de imágenes que se enlistan a continuación:

- **Color jitter**: Esta técnica consiste en alterar las propiedades asociadas a los colores de cada imagen como lo son el brillo, la saturación y el contraste. También permite crear combinaciones de valores para dichas propiedades, lo que da pauta a generar varios resultados sobre la misma imagen.
- **Blur**: También conocido como desvanecimiento, esta técnica aplica un *filtro gaussiano* sobre la imagen, provocando que los valores de los pixeles disminuyan y la imagen original se vea atenuada. Los valores del filtro son inversamente proporcionales al nivel de desvanecimiento, por lo que un mayor nivel de desvanecimiento implica que los valores del filtro son menores y viceversa.
- **Pixelization**: Esta técnica consiste en aprovechar la interpolación de pixeles cuando se realiza un reescalamiento de las imágenes. La primera operación que se realiza con esta técnica es disminuir el tamaño de la imagen, por lo que se requieren menos pixeles para representar la información. La segunda operación es reescalar la imagen a su tamaño original, por lo que es necesario realizar una interpolación de los valores de los pixeles para cubrir los valores requeridos en el nuevo tamaño, lo que se traduce en información artificial que genera la pixelización de las imágenes.
- **Encoding quality**: También conocida como nivel de compresión, la calidad de codificación se refiere al nivel de pérdida de información que recibe una imagen para poder representar sus valores en un formato entendible. Es importante recordar que, para todas las imágenes usadas en este proyecto, el formato de codificación usado es JPEG, por lo que alterar el nivel de compresión de la imagen implica que su calidad se vea alterada. El resultado puede observarse como pérdida de pixeles que impiden que la imagen se muestre con claridad o se muestre borrosa.

- **Shuffle pixels**: Esta técnica consiste en intercambiar, de forma aleatoria, los valores de los pixeles de una imagen. Basado en un factor de intercambio, los valores de los pixeles modifican su posición internamente, lo que visualmente puede interpretarse como ruido no deseado sobre las imágenes. El factor de intercambio puede entenderse como el porcentaje total de pixeles que debieran ser intercambiados, por lo que un valor alto del factor de intercambio implica una mayor cantidad de pixeles a ser intercambiados.

Las técnicas anteriores fueron aplicadas de forma individual y combinadas, por lo que, al usar dos o más técnicas en conjunto, se generaron imágenes con mayor variabilidad gracias a las alteraciones que sufrieron. Algunos de los resultados obtenidos por realizar las operaciones anteriores pueden observarse en las Figuras 5.12, 5.13 y 5.14.

Figura 5.12: Resultado de aplicar las técnicas *blur* y *shuffle pixels*.

Figura 5.13: Resultado de aplicar la técnica *encoding quality*.

Figura 5.14: Resultado de aplicar las técnicas *jitter* y *pixelization*.

Capítulo 6

MODELADO DE LA ESTRUCTURA DEL ROBOT

Continuando con la construcción de la solución que plantea la metodología en cascada, la siguiente tarea consiste en la contrucción de un sistema robótico que cumpla con los objetivos planteados en el apartado *Análisis del problema*. Anteriormente en el apartado *Diseño general de la solución* se analizaron distintas opciones de robots acuáticos, los cuales fueron desarrollados para realizar tareas similares a las tareas establecidas en este proyecto. También se plantearon posibles mejoras que pudieran tener dichos diseños y los materiales requeridos para su implementación.

El objetivo de este capítulo es describir el proceso que se siguió para diseñar la estructura del robot, el cual es uno de los módulos que componen el diseño de la solución propuesta. También se incluyen modelos en 3D de los elementos que conforman la estructura y una explicación del proceso de manufactura seguido para la creación de algunos de ellos. Por último, se muestra la integración de los elementos manufacturados con la intención de visualizar el prototipo final construido con dichos elementos.

6.1. Propuesta de estructura del robot

De acuerdo con el apartado *Diseño general de la solución*, los diseños de robot que fueron presentados poseen características que los hacen una opción viable para ser implementados en este proyecto. Sin embargo, debido a que su propósito inicial está orientado a resolver otras problemáticas, cuentan con algunas desventajas que impiden que sean adaptados por completo a la solución propuesta. Ante ello, se planteó una serie de mejoras con la intención de generar un diseño propio que atienda todas las problemáticas expuestas en este trabajo.

Las mejoras a realizar sobre ambas propuestas abarcan el uso correcto de materiales para la estructura, el diseño de un sistema de flotación auxiliar, y una buena distribución de los componentes electrónicos. Si bien las mejoras anteriores son relevantes, un buen diseño de la estructura del robot que le permita cumplir con sus funciones es el objetivo principal. Para ello, se tiene contemplado usar una estructura similar al robot anfibio [11], la cual pueda flotar sobre la superficie acuática sin complicaciones. Por otro lado, se tiene contemplado tomar la distribución de los componentes del robot pescado [40] y adaptarlos a los componentes a usar en esta solución, lo que permitirá que el robot cumpla con sus funciones sin alterar gravemente las características físicas del mismo.

El diseño del robot debe considerar el uso de un motor que proporcione la fuerza necesaria para mover la estructura, además de incluir espacio suficiente para colocar todos los componentes electrónicos siguiendo una buena distribución. El diseño también debe incluir un sistema de flotación auxiliar que debe ser construido considerando los materiales disponibles, además de ser compatible con la estructura del robot. Por último, debe permitir que la cámara encargada de capturar las fotografías tenga el ángulo correcto para obtener las imágenes, además de proporcionar la suficiente iluminación para que las fotografías puedan capturarse con la mejor claridad posible.

Los elementos que se presentan a continuación fueron desarrollados con la intención de cumplir con los requerimientos planteados para el sistema robótico, además de atender las mejoras propuestas y alinearse a los materiales disponibles para su desarrollo.

6.2. Modelo 3D

Tomando en cuenta que el diseño del robot desarrollado (de ahora en adelante *Robot propulsado*) está basado en el diseño del robot anfibio [11], la estructura final consta de 7 partes únicas que fueron modeladas en 3D usando el software *NX 12* [63]. También se crearon planos de las piezas con la intención de mostrar las medidas de cada una de ellas, además de proporcionar distintas vistas con el proposito de facilitar la replicabilidad de la estructura en un futuro.

La primera parte única es la **base del robot**, la cual cuenta con un diseño rectangular que permite que los componentes electrónicos sean montados sobre ella. El modelo 3D de la pieza puede observarse en la Figura 6.1, mientras que su plano correspondiente puede observarse en el Apéndice C con la etiqueta C.1.

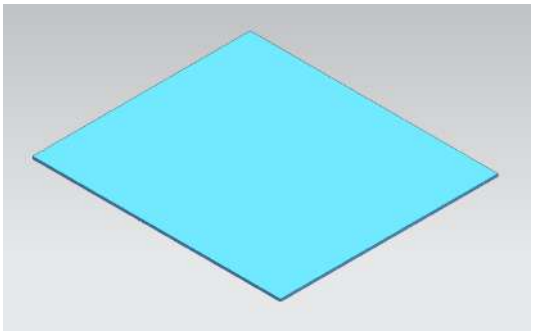


Figura 6.1: Modelo 3D de la pieza *base* de Robot propulsado.

La siguiente parte única corresponde a la **pared frontal inferior**, la cual debe ser ensamblada con un ángulo de 37° con respecto a la base. La intención de esta pieza es facilitar la toma de fotografías bajo el agua y soportar la cámara que estará encargada de hacerlo, así como sostener los elementos externos que provean la iluminación suficiente para la captura de las imágenes. El modelo 3D puede observarse en la Figura 6.2, mientras que el plano con las medidas de la pieza se muestra en el Apéndice C con la etiqueta C.2.

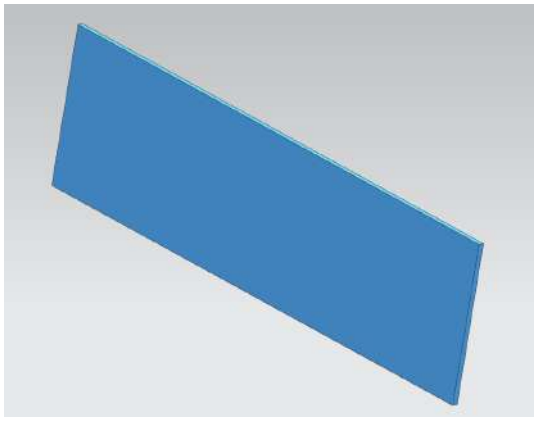


Figura 6.2: Modelo 3D de la pieza *pared frontal inferior* de Robot propulsado.

La tercera parte única corresponde a la **pared trasera inferior**, la cual debe ser ensamblada con un ángulo de 45° con respecto a la base. La intención de esta pieza es mantener la geometría del robot, así como facilitar la navegación de la estructura completa en conjunto con el sistema de flotación auxiliar. El modelo 3D de la pieza puede observarse en la Figura 6.3, mientras que su plano correspondiente puede obsrevarse en el Apéndice C bajo la etiqueta C.3.

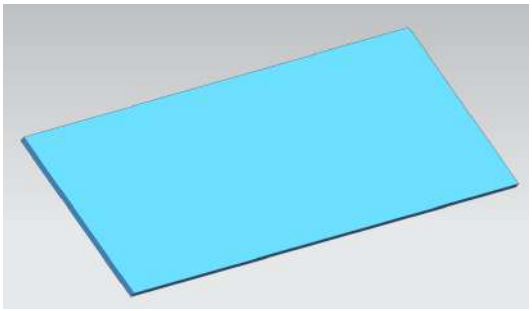


Figura 6.3: Modelo 3D de la pieza *pared trasera inferior* de Robot propulsado.

Otra pieza única usada en la estructura del robot es el **soporte del sistema de flotación**, el cual fue definido como un cilindro montado en múltiples ocasiones a lo largo de las paredes de la estructura. El propósito de incluir estos elementos es que sean unidos al sistema de flotación manteniendo una distancia constante con respecto al nivel del agua, además de evitar que el sistema de flotación se separe de la estructura. Los soportes se encuentran perforados por un costado con la intención de reducir el peso de cada uno de ellos, mientras que el otro costado se encuentra totalmente sellado para evitar filtraciones de agua. El modelo 3D de la pieza se muestra en la Figura 6.4, mientras que su plano correspondiente puede observarse en el Apéndice C bajo la etiqueta C.4.

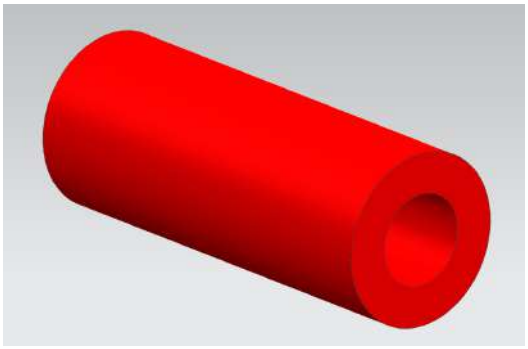


Figura 6.4: Modelo 3D de la pieza *soporte* de Robot propulsado.

La quinta parte única que compone la estructura del robot es la **pared trasera superior**, la cual fue diseñada como un rectángulo con una perforación en su interior. El propósito de la perforación en la pieza es permitir la conexión entre el motor y los elementos que lleven a cabo la propulsión de la estructura, por lo que esta pieza debe ser ensamblada verticalmente para permitir que los componentes para la propulsión estén en contacto con el agua a un nivel correcto. El modelo 3D de la pieza puede observarse en la Figura 6.5, mientras que su plano correspondiente puede observarse en el Apéndice C bajo la etiqueta C.5.

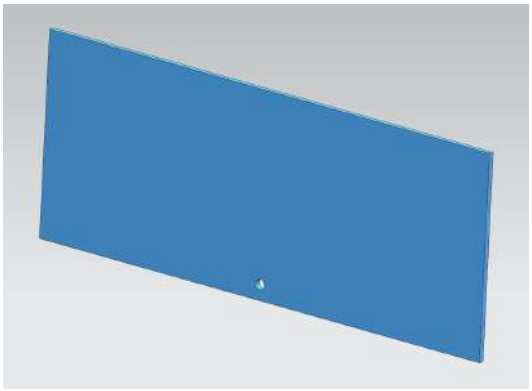


Figura 6.5: Modelo 3D de la pieza *pared trasera superior* de Robot propulsado.

La penúltima parte única de la estructura es la **pared frontal superior**, la cual fue definida como un rectángulo con dos circulos en su interior distribuidos de forma simétrica. Los circulos en la pieza permiten que los soportes del sistema de flotación puedan ser montados a ella, de tal manera que, al integrar el sistema de flotación auxiliar, sea el mismo sistema el que pueda proteger la estructura ante posibles objetos que la pudieran dañar mientras navega. El modelo 3D de la pieza puede observarse en la Figura 6.6, mientras que el plano con las medidas de la pieza se puede observar dentro del Apéndice C bajo la etiqueta C.6.

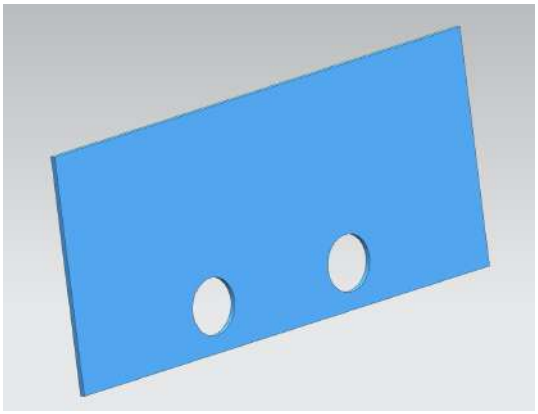


Figura 6.6: Modelo 3D de la pieza *pared frontal superior* de Robot propulsado.

La última parte única que compone la estructura del robot es la **pared lateral**, la cual es la pieza más grande de todo el diseño. Esta pieza fue definida como un hexágono irregular con tres círculos en su interior distribuidos de forma simétrica. Al igual que la pared frontal superior, los circulos en la pieza permiten que los soportes del sistema de flotación puedan ser montados a ella, logrando que el sistema de flotación auxiliar rodee la estructura y se facilite la flotación de todo el robot. Esta pieza es usada en dos ocasiones, por lo que cada una debe tener la presencia de sus respectivos soportes. El modelo 3D de la pieza puede observarse en la Figura 6.7, mientras que su plano correspondiente puede observarse en el Apéndice C bajo la etiqueta C.7.

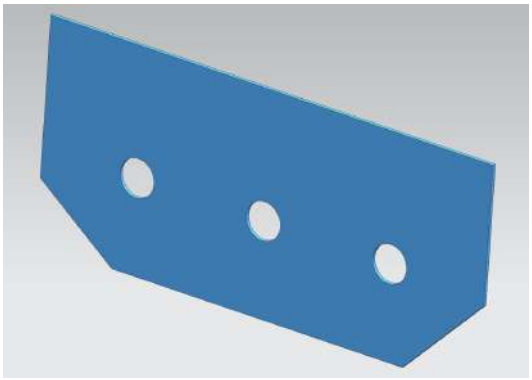


Figura 6.7: Modelo 3D de la pieza *pared lateral* de Robot propulsado.

Como se mencionó al inicio de esta sección, las partes únicas que fueron presentadas anteriormente son las piezas que conforman la estructura final del robot. En total, la estructura está compuesta por 15 piezas, de las cuales 8 corresponden a los soportes para el sistema de flotación y 2 a las paredes laterales. Para simular la integración de todas las piezas se usó nuevamente el software NX 12, el cual permite el ensamble de las piezas modeladas en 3D considerando las restricciones y medidas indicadas. Las Figuras 6.8 y 6.9 muestran la integración final de las piezas del robot, además de contar con el sistema de flotación auxiliar añadido a los soportes. El plano de la integración puede observarse en el Apéndice C bajo la etiqueta C.8.

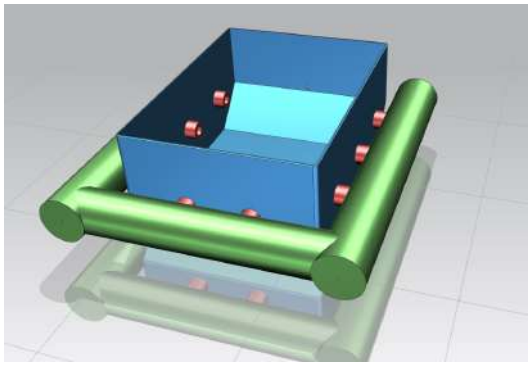


Figura 6.8: Modelo 3D de Robot propulsado: vista delantera.

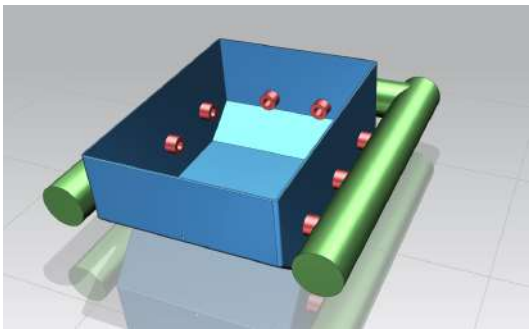


Figura 6.9: Modelo 3D de Robot propulsado: vista trasera.

6.3. Manufactura de los componentes

Una vez que se modelaron los componentes de la estructura del robot, la siguiente tarea fue la manufactura de cada elemento, por lo que se debieron tomar en cuenta los materiales requeridos para la estructura y la maquinaria disponible para realizarlos. En este paso se usaron dos métodos de manufactura distintos, los cuales serán presentados a continuación con la intención de conocer el proceso que se siguió en cada pieza.

6.3.1. Corte mecanizado con láser

La manufactura con corte láser fue empleada para crear las paredes de la estructura del robot usando láminas de polimetilmetacrilato. La intención de emplear este tipo de manufactura es poder realizar operaciones de corte precisas, además de facilitar los procesos de acabado de las perforaciones que requieren algunas de las piezas. Debido a que el polimetilmetacrilato es un material que puede ser moldeado y perforado a muy altas temperaturas, la manufactura con corte láser resulta la mejor alternativa para elaborar cada una de las paredes.

Para llevar a cabo este proceso, se convirtió cada uno de los modelos en 3D a un archivo de vectores que permitiera identificar los contornos de cada pieza. Posteriormente, se solicitó a un proveedor externo la manufactura de ellas en láminas de polimetilmetacrilato de 3 mm de grosor, facilitando los procesos de corte y de acabado de las perforaciones gracias al grosor del material. Un ejemplo del proceso elaborado para la manufactura con corte láser puede observarse en las Figuras 6.10 y 6.11, donde la primera figura muestra el archivo de vectores generado para la pared lateral, mientras que la segunda figura muestra el resultado después de la manufactura de la pieza.

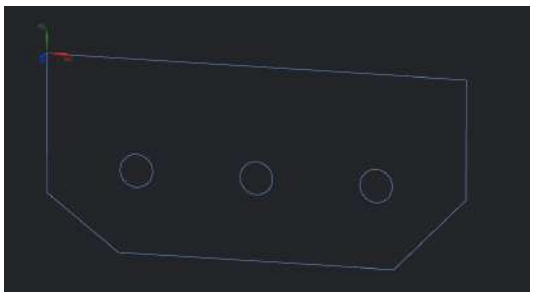


Figura 6.10: Pared lateral en archivo de vectores.

Figura 6.11: Pared lateral manufacturada por corte láser.

6.3.2. Manufactura de maquinado por control numérico

Este tipo de manufactura fue empleada para la creación de los soportes del sistema de flotación auxiliar, elaborando múltiples elementos de forma rápida con ayuda de las máquinas de manufactura de los laboratorios del ITAM. Debido a que la estructura del robot requiere 8 soportes cilíndricos con las mismas dimensiones y características, se tomó la decisión de elaborarlos con ayuda de un torno por control numérico (*CNC*), permitiendo manufacturar las piezas con una única serie de instrucciones.

Como puede observarse en el plano C.4, los soportes poseen una dimensión cilíndrica regular de 27 mm de diámetro, además de tener una perforación interna de 14.3 mm de diámetro con la intención de reducir el peso de cada uno de ellos. Para generar la serie de instrucciones correspondientes (de ahora en adelante llamadas *Código-G*), se requiere indicar las tareas que debe realizar el torno con las herramientas disponibles, además de tener en cuenta el material con el que se está trabajando y las características físicas de la máquina. Con la intención de facilitar la generación del código-G, se usó el módulo *Manufacturing* del software NX 12, el cual provee una simulación del proceso de manufactura usando máquinas por control numérico, además de ayudar a generar el código-G basado en las operaciones simuladas.

El primer paso de la simulación fue la configuración de las variables del espacio de trabajo de la máquina, las cuales permiten definir las zonas para cambio de herramientas y los límites de seguridad con el fin de evitar un choque entre el eje principal del torno y la torreta de las herramientas. El segundo paso

consistió en analizar la pieza a manufacturar y el material a usar, por lo que se establecieron ciertas medidas iniciales para que cada operación de manufactura pudiera ser realizada con la mejor calidad posible. El tercer paso en la simulación fue la elección y definición de las herramientas, las cuales varían según sean las operaciones a realizar y el material a usar para la manufactura.

A continuación se presenta una serie de tablas que resumen la información necesaria para la manufactura de los soportes. Las Tablas 6.1, 6.2 y 6.3 proporcionan información sobre la definición de las herramientas usadas, mientras que la Tabla 6.4 proporciona los valores de las propiedades para realizar cada operación del proceso de manufactura.

Herramientas de desbaste					
Herramienta	Forma	Punta	Radio	Orientación	Tamaño
OD_80_L	Diamante	80.0°	1.2 mm	5°	15.0 mm
OD_35_L	Diamante	35.0°	0.2 mm	52°	10.0 mm

Tabla 6.1: Características de las herramientas de desbaste.

Broca para perforación					
Diámetro	Tamaño	Apertura	Largo apertura	Esquina	Hélice
14.288 mm	80.0 mm	118.0°	4.2925 mm	0.0°	35.0 mm

Tabla 6.2: Características de la broca para perforar.

Herramienta de corte					
Orientación	Largo	Ancho	Radio	Ángulo lateral	Punta
90°	5.2 mm	3.0 mm	0.0 mm	0.0°	0.0°

Tabla 6.3: Características de la herramienta de corte.

Operaciones de manufactura					
Propiedad \Operación	Careado	Perforación	Desbaste	Acabado	Corte
Dirección	*Forward*	*Forward*	*Forward*	*Forward*	*Forward*
Profundidad max.	1.0 mm	55.0 mm	2.0 mm	———	———
Profundidad min.	0.0 mm	0.0 mm	0.0 mm	0.0 mm	———
Velocidad (SMM)	150.0	120.0	150.0	40.0	———
Max RPM	1500.0	2673.0	1500.0	400.0	150.0
Corte (mmpr)	0.05	0.125	0.05	0.05	0.7
Feed per tooth	———	0.0625	———	———	———

Tabla 6.4: Valores de las operaciones de manufactura.

Las operaciones mostradas en la Tabla 6.4 son las operaciones elegidas para manufacturar los soportes. Los valores y propiedades que se observan en ella son los valores requeridos para trabajar de forma correcta con Nylamid, material elegido gracias a su compatibilidad con las herramientas disponibles en los laboratorios del ITAM. A continuación se enlistan, en orden de uso, todas las operaciones del proceso de manufactura elaborado, así como el resultado esperado de cada una de ellas.

1. **Careado o *Facing***: es una operación que permite realizar un primer desbaste del material con la intención de indicar el inicio de la pieza manufacturada. También permite eliminar impurezas del material o posibles sobrantes. El resultado esperado puede observarse en la Figura 6.12, la cual muestra con color amarillo el material a usar, con color café la pieza a generar y con líneas verde-azules los cortes realizados por la herramienta.

2. **Perforado o *Drilling***: esta operación usa una broca para perforar el material con base en una profundidad objectivo, en este caso la profundidad del soporte del sistema de flotación. El resultado esperado puede observarse en la Figura 6.13, donde al fondo de la pieza se muestra con color amarillo la punta de la broca a usar, lo que indica que la pieza tendrá esa geometría después de la perforación.

3. **Desbaste o *Roughing***: en esta operación se realiza un desbaste del material innecesario para la creación de la pieza, por lo que en esta operación se debe cumplir con los límites del área de trabajo para evitar un choque entre la herramienta de corte y el eje principal. El resultado esperado puede observarse en la Figura 6.14, la cual muestra con líneas verde-azules los cortes realizados por la herramienta, mientas que con una línea naranja punteada se indica el límite permitido para cortar.

4. **Acabado o *Fishing***: es una operación que permite eliminar el material sobrante y dar forma a la pieza en curso. Permite eliminar impurezas del material y definir el contorno de la pieza. Al igual que el desbaste, esta operación requiere cumplir con los límites del área de trabajo para evitar un choque. El resultado esperado puede observarse en la Figura 6.15, la cual muestra con color azul la región de material eliminada, con una línea naranja punteada el limite permitido para cortar y con color café la pieza objetivo a elaborar.

5. **Corte o *Part off***: esta última operación del proceso tiene la intención de retirar las piezas una vez que han sido manufacturadas. Permite cortar verticalmente la base de la pieza para mantener intacta la estructura. El resultado esperado puede observarse en la Figura 6.16, la cual muestra con una línea verde-azul el punto de corte del material para ser retirado de la máquina.

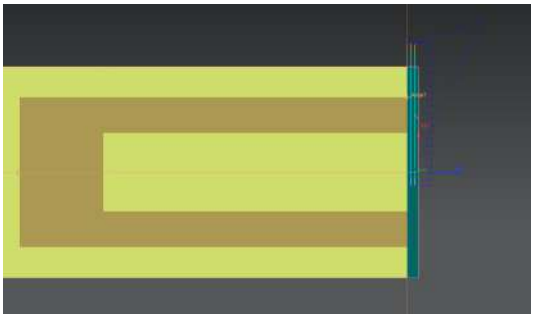


Figura 6.12: Resultado esperado después del careado del material.

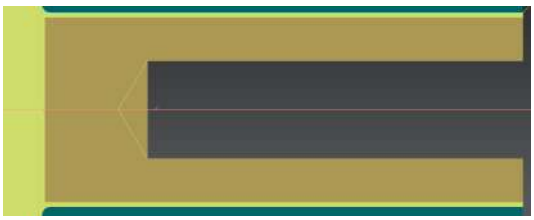


Figura 6.13: Resultado esperado después de la perforación del material.

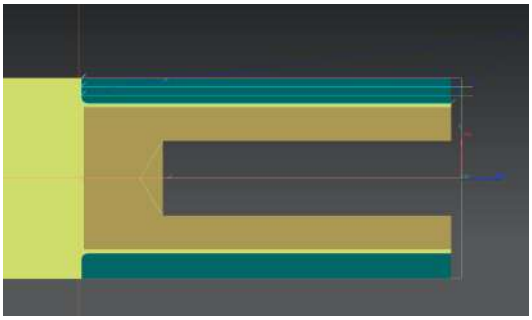


Figura 6.14: Resultado esperado después del desbaste del material.

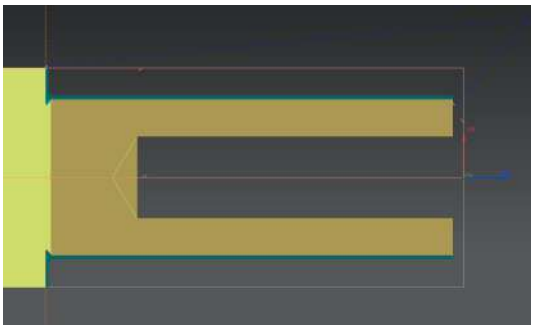


Figura 6.15: Resultado esperado después del acabado del material.

Figura 6.16: Resultado esperado después del corte del material.

Una vez finalizada la simulación de las operaciones del proceso de manufactura, el último paso fue generar el código-G con ayuda de un post-procesador. Para este trabajo, el post-procesador que se usó fue el proporcionado en la clase *Manufactura integrada por computadora* del ITAM, el cual genera la documentación y las tareas a realizar por el torno para cada operación. Los códigos-G pueden observarse en el Apéndice D, donde el Código D.1 es el encargado de invocar cada una de las operaciones a realizar, mientras que los Códigos D.2, D.3, D.4, D.5 y D.6 corresponden a las opreaciones de Careado, Perforado, Desbaste, Acabado y Corte respectivamente.

La Figura 6.17 muestra la evidencia del proceso de manufactura de los soportes para el sistema de flotación auxiliar, en la cual se puede observar al torno CNC de los laboratorios del ITAM realizando la operación de desbaste del material con ayuda del código-G generado.

Figura 6.17: Manufactura en torno CNC de los soportes.

6.3.3. Integración física de los componentes

Después de manufacturar los elementos que conforman la estructura del robot, la última actividad fue la integración de todos los componentes. Como se mencionó en el apartado *Diseño general de la solución*, la unión de las piezas se realizó con pegamento y plastilina epóxicos con el fin de evitar la filtración de agua y asegurar una buena fijación entre las piezas. La Figura 6.18 muestra la estructura del robot flotando sobre una superficie acuática después de la integración de sus componentes.

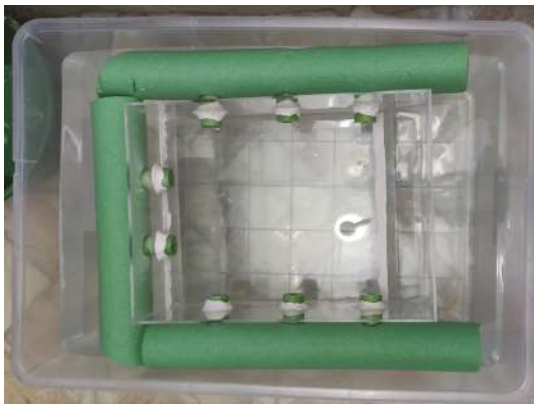


Figura 6.18: Estructura del robot flotando sobre una superficie acuática.

Capítulo 7

IMPLEMENTACIÓN DE LA SOLUCIÓN

El último paso a seguir dentro de la etapa de la construcción de la solución es la implementación e integración de cada uno de los elementos que la componen. Tomando en cuenta el trabajo previo realizado sobre la preparación de las imágenes para el modelo de detección de objetos, así como la manufactura y construcción del sistema robótico, se requieren algunas tareas adicionales que permitan que la solución sea un prototipo completo y robusto, el cual pueda cumplir con los objetivos y alcances definidos al inicio de este trabajo.

Continuando con el Proceso de Vijay Kotu [36] expuesto en el apartado de *Recolección y preparación de las imágenes*, una vez que fueron preparadas las muestras de los datos a usar, se puede proceder a desarrollar un modelo de aprendizaje de máquina capaz de resolver el problema encontrado. De igual forma, cuando se presentó el diseño del módulo de visión en el apartado *Diseño general de la solución*, se mostraron los componentes a usar para el desarrollo de dicho módulo, destacando el uso de arquitecturas definidas de redes neuronales para facilitar la implementación del modelo de detección de objetos. Por lo tanto, en este capítulo se incluye una explicación del proceso realizado para la creación de varios modelos de detección de objetos, así como las herramientas usadas para favorecer su implementación.

Por otro lado, en el apartado *Diseño de la estructura del robot* se detalló el proceso de construcción de la estructura del sistema robótico, dejando de lado la integración de los componentes electrónicos y periféricos propuestos para este módulo. Adicionalmente, en el apartado *Diseño general de la solución* se presentó un diagrama (Figura 4.1) con la intención de ejemplificar visualmente la composición de la solución propuesta, denotando la necesidad de contener todos los elementos de la solución dentro de la estructura del sistema robótico. Es por ello que en este capítulo también se presenta la integración y distribución de cada uno de los elementos que componen al sistema robótico, además de proporcionar recursos visuales que permitan entender las conexiones electrónicas y ejemplifiquen la posición de todos los elementos usados.

7.1. Desarrollo y entrenamiento de los modelos de detección de objetos

De acuerdo con el proceso documentado en el apartado *Recolección y preparación de las imágenes*, se obtuvieron 2040 imágenes del dataset WaDaBa que fueron etiquetadas de acuerdo con la clase de plástico del objeto que estuviera presente en ellas, siendo las clases de plástico *PET*, *PS* y *PEHD* las clases elegidas a detectar. Así mismo, se recolectaron 2040 imágenes complementarias que no poseen algún tipo de plástico de las clases a detectar, por lo que se usó la información de los objetos presentes en dichas imágenes para aumentar la precisión de los modelos desarrollados. Adicionalmente, se llevó a cabo una generación artificial de imágenes basadas en ciertas técnicas de procesamiento y filtros para imágenes, permitiendo incrementar la cantidad de información disponible y diversificar las características de cada imagen obtenida.

Cabe recordar que se obtuvieron 4080 archivos de texto asociados a las imágenes recolectadas, es decir 2040 archivos de texto con la información de los objetos plásticos que nos interesa detectar, y 2040 archivos en blanco para las imágenes sin objetos plásticos de interés. Para las imágenes generadas artificialmente también se generó su archivo de texto correspondiente, sin embargo la cantidad de imágenes artificiales fue diversa según el modelo desarrollado.

7.1.1. Prerrequisitos para el entrenamiento de los modelos

Como se planteó en el apartado *Diseño general de la solución*, se optó por generar diversos modelos de las arquitecturas YOLOv3 y YOLOv4 con la intención de comparar su desempeño y evaluar su precisión a la hora de detectar los objetos plásticos. Ambas arquitecturas cuentan con características similares, ya que son el resultado de mejorar ciertas capacidades de versiones pasadas, sin embargo es necesario valorar cuál es la arquitectura que mejor se ajusta a las necesidades del problema y nos permite alcanzar los resultados esperados.

Siguiendo las fases para el desarrollo de modelos de aprendizaje de máquina expuestas en la Figura 3.3, la primera fase que se abordó fue la del entrenamiento de los modelos. Para entrenar algún modelo de detección de objetos usando las arquitecturas YOLO se requiere de un *framework* que sea compatible y que estandarice el proceso del entrenamiento de redes neuronales, por lo que se decidió usar el framework **Darknet** [53] debido a que su funcionamiento está optimizado para el entrenamiento de modelos usando dichas arquitecturas. Darknet también brinda la capacidad de acelerar el entrenamiento de los modelos mediante el uso de **Unidades de procesamiento gráfico** (*GPU* por sus siglas en inglés), lo que permite reducir el tiempo que tomaría generar varios modelos y permite destinar más tiempo a la evaluación de cada uno de ellos.

A pesar de la restricción de contar únicamente con una computadora para trabajar, esta computadora posee una tarjeta GPU integrada que permitió acelerar el entrenamiento de los modelos desarrollados usando Darknet. Buscando aprovechar este recurso, se tuvo que seguir una serie de instrucciones previas[1] para configurar y definir la comunicación entre Darknet y el modelo de la tarjeta GPU que se posee. En conjunto con el entrenamiento de los modelos en la computadora disponible, se creó un recurso de acceso libre desarrollado en Google Colab, el cual permite el entrenamiento de modelos de detección de objetos haciendo uso de las tarjetas GPU que provee Google Cloud. Este recurso se describe en el apartado *Aportaciones adicionales* bajo la etiqueta 10.2.

[1]Liga para consultar la configuración realizada y los requisitos de instalación: https://medium.com/geekculture/install-cuda-and-cudnn-on-windows-linux-52d1501a8805

7.1.2. Conjuntos de entrenamiento y validación

Retomando la información presentada en el apartado *Comprensión teórica*, las redes neuronales son un método de aprendizaje de máquina basado en el paradigma del aprendizaje supervisado, para el cual es necesario proporcionar información desconocida a la entrada del modelo y, mediante una retroalimentación, se hace una corrección de los errores cometidos con la intención de afinar su desempeño. En este proyecto, la información proporcionada durante la fase del entrenamiento de los modelos fue tomada a partir de conjuntos de imágenes de objetos plásticos y de las imágenes adicionales recolectadas. Se crearon tres conjuntos distintos de datos, los cuales se enlistan a continuación:

- **Conjunto 1**: 2040 imágenes con objetos plásticos.
- **Conjunto 2**: 2040 imágenes con objetos plásticos, y 4080 imágenes artificiales generadas a partir de las imágenes adicionales con la combinación de las técnicas *blur*, *color jitter* y *encoding quality* (6120 imágenes en total).
- **Conjunto 3**: 2040 imágenes con objetos plásticos, 2040 imágenes adicionales, y 34722 imágenes artificiales generadas con la combinación de todas las técnicas de data augmentation presentadas (38802 imágenes en total).

Para cada uno de los conjuntos anteriores se realizó un **muestreo aleatorio** seleccionando el 93 % del total de las imágenes para conformar el subconjunto de imágenes de entrenamiento, mientras que el 7 % de imágenes restantes conformaron el subconjunto de validación. La proporción elegida para realizar la división de las imágenes fue seleccionada con base en la cantidad de imágenes con objetos plásticos que se tenían disponibles, además de seguir un proceso iterativo donde se probaron varias proporciones hasta encontrar la proporción que ayudara a generalizar el problema, así como poder evaluar el desempeño del modelo con una cantidad de imágenes suficiente.

El muestreo aleatorio fue llevado a cabo con ayuda de un código en Python que puede ser consultado en el Apéndice A bajo la etiqueta A.3. Este código realiza la división de los subconjuntos de entrenamiento y validación, crea un compilado de las ubicaciones de las imágenes que conforman cada subconjunto, y genera los archivos que requiere Darknet para realizar el entrenamiento

de algún modelo. Es importante destacar que los subconjuntos de validación son requeridos por cada modelo entrenado para evaluar su desempeño de forma automática, además de que le permiten al usuario poder identificar errores durante el proceso de entrenamiento con ayuda de las evaluaciones y las gráficas de pérdida generadas.

7.1.3. Entrenamiento y generación de los modelos

A partir de los subconjuntos de entrenamiento y validación creados, se generaron seis modelos distintos de detección de objetos: tres modelos con la arquitectura YOLOv3, y tres modelos con la arquitectura YOLOv4. Para cada una de las arquitecturas se definieron algunos valores de *hiperparámetros*, los cuales son valores asociados a cada modelo de redes neuronales que ayudan a definir su comportamiento. Así mismo, se requirió definir algunos valores de configuración para Darknet con el propósito de ejecutar el entrenamiento de los modelos. La tabla 7.1 contiene un resumen de los valores de los hiperparámetros y de configuración definidos para cada arquitectura usada:

Valores de hiperparámetros y configuración		
Valor \Arquitectura	YOLOv3	YOLOv4
Batch	64	64
Subdivisions	32	64
Max. batches	8000	8000
Steps	6400 - 7200	6400 - 7200
Width	416	416
Height	416	416
Learning rate	0.001	0.001
Scales	0.1 - 0.1	0.1 - 0.1
Momentum	0.9	0.949
Classes	3	3
Filters for YOLO layers	24	24
Activation function	Leaky	Mish

Tabla 7.1: Valores de hiperparámetros y configuración definidos.

Cada modelo fue entrenado durante 8000 iteraciones (*Max. batches*), donde en cada iteración se tomó una muestra del subconjunto de imágenes de entrenamiento (*Batches*), se generaron n divisiones de la misma cantidad de imágenes (*Subdivisions*), y se procesó cada división de forma paralela. Antes de ser proporcionadas a Darknet, cada imagen del subconjunto de imágenes de entrenamiento fue reescalada a un tamaño que pudiera ser fácil de procesar (*Width* y *Height*), además de cumplir con los requerimientos de cada arquitectura para poder llevar a cabo el entrenamiento.

Con la intención de afinar el valor de los pesos generados en cada iteración del entrenamiento de los modelos, se definieron iteraciones clave (*Steps*), en las cuales la fuerza de aprendizaje (*Learning rate*) sería disminuida en un factor (*Scales*) para poder encontrar algún valor de pérdida mínimo. Así mismo, para hacer más sencilla la tarea de encontrar dicho valor de pérdida mínimo, se definió un coeficiente (*Momentum*) que hiciera más ligeros los cálculos del valor de pérdida por cada iteración realizada, considerando el valor de pérdida de la iteración anterior.

Finalmente, fue necesario indicar a Darknet cuántas clases de plástico debían ser detectadas (*Classes*), la cantidad de filtros en las capas finales para realizar la clasificación (*Filters for YOLO layers*), y la función matemática a usar entre capas (*Activation function*) para escalar los valores obtenidos y facilitar la detección de los objetos. Usando los valores de hiperparámetros y de configuración anteriores, el tiempo promedio que tomó entrenar cada uno de los modelos fue de 25 horas, considerando que el tiempo de entrenamiento se ve impactado por la cantidad de capas que componen cada arquitectura, la cantidad de iteraciones (*Max. batches*), y las condiciones del hardware de la computadora usada.

7.2. Desarrollo del sistema robótico

Previamente en el apartado *Diseño de la estructura del robot* se describieron las tareas realizadas para manufacturar y construir la estructura del sistema robótico. Tal como se explicó en dicho apartado, el diseño propuesto tiene la intención de cumplir con todos los requerimientos planteados para este proyecto,

así como contener a todos los elementos electrónicos y periféricos descritos en el apartado *Diseño general de la solución*. Adicionalmente, se presentó evidencia de la estructura del robot en un ambiente acuático casero, donde se puede observar que el objetivo de flotar de forma autónoma sin poseer filtraciones de agua o hundimientos parciales se cumplió.

Después de tener una estructura sólida y funcional probada con éxito, la siguiente tarea fue realizar la distribución de los componentes electrónicos y periféricos considerados para la solución. Esta tarea fue muy importante debido a que una correcta distribución de los componentes ayuda a reducir problemas de hundimiento por concentración de peso en algunas zonas, además de facilitar la navegación del sistema sobre la superficie acuática, y asignar un lugar específico a cada elemento para fácil identificación de problemas.

7.2.1. Integración de elementos electrónicos

De acuerdo con la información presentada en el apartado *Diseño general de la solución* referente a los elementos electrónicos a usar, cada uno de estos elementos tiene un propósito específico, desde brindar suficiente luz a la cámara fotográfica en uso, hasta ofrecer el movimiento necesario al robot para navegar sobre la superficie acuática. Teniendo en cuenta la funcionalidad de cada componente, se requirió una distribución estratégica que pudiera ser replicada y que fuera de fácil acceso en caso de requerir alguna modificación, además de nivelar el peso de la estructura completa para evitar posibles hundimientos.

Los primeros elementos integrados fueron los sensores, los cuales son los encargados de evaluar si la iluminación del ambiente navegado es suficiente para capturar imágenes con buena visibilidad. Para cumplir con esta tarea, los sensores fueron ubicados en conjunto con un par de luces LED en la pared frontal inferior de la estructura del robot, permitiendo evaluar constantemente si la luz ambiental es suficiente o si se debiera proporcionar iluminación externa. La Figura 7.1 muestra la ubicación de los sensores y las luces LED integrados en la estructura del robot, donde se puede observar que están distribuidos de forma simétrica a un costado de cada uno de los soportes del sistema de flotación, lo que permite recabar suficiente información sobre la iluminación ambiental y proporcionar mayor iluminación externa a la cámara.

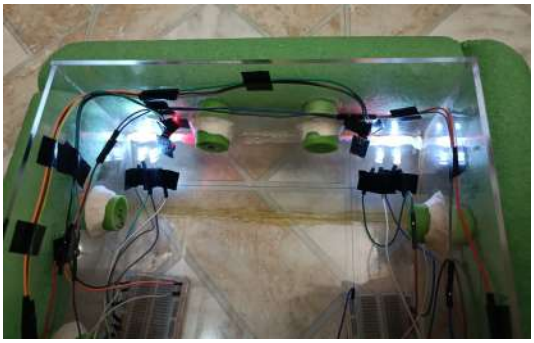


Figura 7.1: Sensores de iluminación y LEDs integrados a la estructura del robot.

El segundo elemento implementado fue el microcontrolador Arduino, ya que es el encargado de realizar las lecturas enviadas por el sensor de iluminación, hacer la evaluación de los datos leídos, e indicar la intensidad de iluminación que las luces LED debieran proporcionar. Para facilitar el envío de señales y de corriente entre elementos electrónicos, se usaron un par de *protoboards* y cables *jumper*, los cuales permiten identificar rápidamente errores en el sistema electrónico y extienden las conexiones entre elementos. Adicionalmente, para suministrar de energía al sistema electrónico, se conectó la batería a una de las protoboards, lo que facilitó la toma de corriente para los elementos que lo requieren. La Figura 7.2 muestra la ubicación de los elementos anteriores, destacando la ubicación de la batería al centro de la estructura y las protoboards en ambos costados, favoreciendo la extensión de las conexiones y la agrupación del cableado.

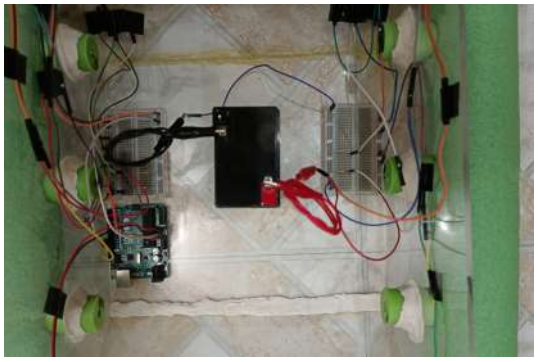


Figura 7.2: Arduino, batería y protoboards integrados a la estructura del robot.

7.2.2. Integración de la microcomputadora y elementos periféricos

Continuando con la integración de los elementos que componen a la solución propuesta, el siguiente elemento que se incluyó fue la cámara para obtener las fotografías del ambiente navegado. Tal como se describió en el apartado *Diseño general de la solución*, para este proyecto se consideraron dos cámaras fotográficas distintas: la cámara GoPro Hero 7 Black, y la cámara integrada del celular Mi 9T, buscando obtener resultados con distintas resoluciones de cámaras y cumplir con el requerimiento de capturar imágenes en tiempo real o lo más parecido a ello. Dado que la estructura del robot está diseñada para ser propulsado desde la parte trasera, la cámara fue ubicada al centro de la pared frontal inferior de la estructura, lo que permite cambiar fácilmente de cámara y favorece la captura de fotografías conforme se navega el ambiente de interés.

Luego de establecer la ubicación de las cámaras, el siguiente elemento integrado fue la microcomputadora Raspberry Pi, la cual es la encargada de realizar todo el procesamiento de las imágenes, ejecutar el algoritmo de detección de objetos sobre cada una de ellas, y visualizar los resultados obtenidos por el algoritmo en un monitor externo. Con la intención de nivelar el peso de la estructura completa, la microcomputadora fue ubicada en la punta de la base del robot, entre la cámara fotográfica y la batería, priorizando la habilitación de los puertos necesarios para energizar la microcomputadora y visualizar su

interfaz gráfica en un monitor externo. De acuerdo con el diseño de la solución, se contemplaron dos tipos de comunicación entre las cámaras fotográficas y la microcomputadora: una alámbrica y otra inalámbrica. Para la comunicación alámbrica bastó realizar una conexión entre los puertos USB de los dispositivos, mientras que para la comunicación inalámbrica se requirió crear una **red de área local)** (LAN por sus siglas en inglés) en donde ambos dispositivos interactuaran mediante sus módulos Wi-Fi. La red fue creada con ayuda de la microcomputadora de forma privada, restringiendo la conexión a todos los dispositivos no autorizados. Así mismo, para facilitar el envío de las imágenes capturadas con el celular hacia la microcomputadora, se usó la aplicación **IP-Webcam** [34], la cual crea un servidor interno en el celular al que se puede acceder dentro de la misma red para obtener las fotografías.

A continuación se muestra un par de imágenes que ilustran la distribución de los componentes mencionados. En la Figura 7.3 se puede observar la ubicación de la cámara GoPro en conjunto con la microcomputadora, mientras que en la Figura 7.4 se puede observar la ubicación del celular (cuya cámara integrada se encuentra en la parte trasera del dispositivo) en conjunto con la microcomputadora.

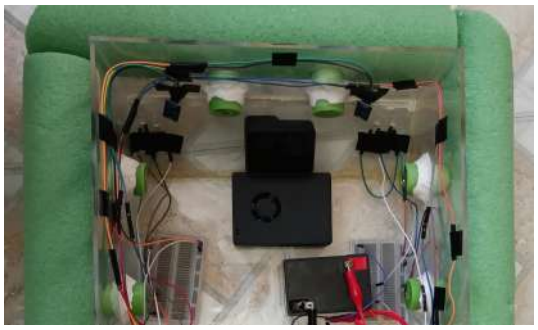


Figura 7.3: Cámara GoPro y Raspberry Pi integrados a la estructura del robot.

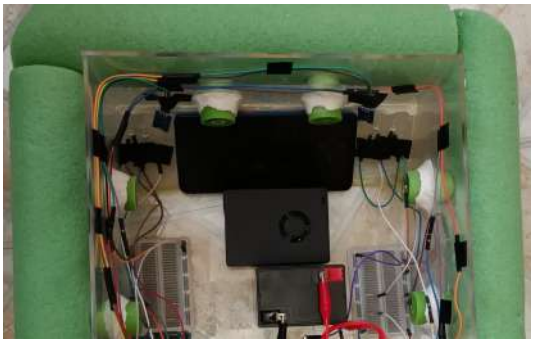


Figura 7.4: Celular Mi 9T y Raspberry Pi integrados a la estructura del robot.

Después de todos los elementos anteriores, el último elemento en ser integrado fue el motorreductor en conjunto con su respectivo controlador. El controlador elegido fue la placa **L293D**, el cual es un controlador con capacidad para dos motores que permite variar la velocidad y el sentido de cada uno de ellos. Debido a que una de las necesidades de este proyecto es utilizar únicamente un motor, el diseño de la placa se ajusta a las necesidades y restricciones de espacio dentro de la estructura del robot, además de contar con un disipador de calor para disminuir la temperatura de los componentes electrónicos que componen a la placa. La Figura 7.5 muestra al controlador L293D con el que se trabajó en este proyecto.

Figura 7.5: Controlador L293D utilizado.

Por otro lado, dado que el diseño del robot propuesto es el de un sistema

propulsado, el motorreductor fue adaptado con un par de elementos complementarios que facilitaran esta labor. En primera instancia se consideró el uso de una hélice que realizara la propulsión junto con el motorreductor, por lo que se optó por usar una hélice de plástico como la que se muestra en la Figura 7.6. En consecuencia, para adaptar la hélice al motor se requirió de un eje de acero y de un cople diseñado e impreso en 3D, creando un sistema de transmisión de velocidad entre el motor y la hélice.

El último problema a resolver fue la altura a la cuál debía estar el motorreductor junto con el eje de acero, teniendo que ajustarse al diseño de la pared trasera superior de la estructura del robot. Para cumplir con este requisito, se creo una base de **madera** con la suficiente altura para poder cruzar el eje de acero por el orificio destinado al sistema de propulsión, además de brindar estabilidad al motorreductor ante vibraciones. En la Figura 7.7 se puede observar la integración del motorreductor con el sistema de propulsión adaptado, los cuales se ubicaron en la punta inferior de la base del robot.

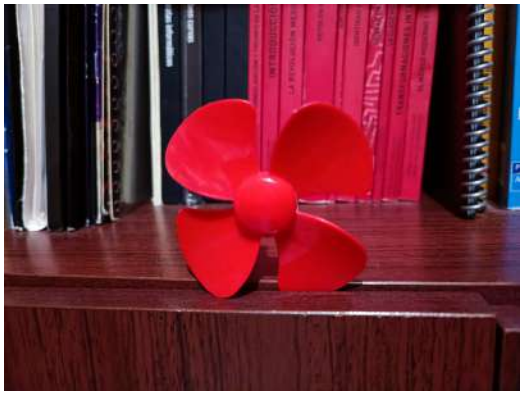


Figura 7.6: Hélice utilizada para realizar la propulsión.

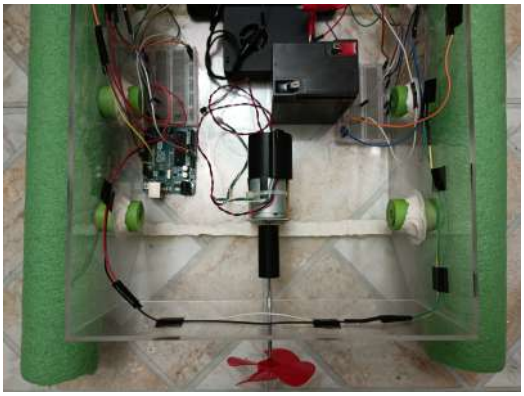


Figura 7.7: Motorreductor y sistema de propulsión adaptados a la estructura del robot.

Como parte de la implementación de la solución propuesta, con ayuda del software **Fritzing** se creó un diagrama de las conexiones electrónicas realizadas con la intención de tener una futura referencia en caso de requerir alguna modificación o mejora, o en su defecto poder replicarlas sencillamente. Dicho diagrama puede ser visualizado en la Figura 7.8.

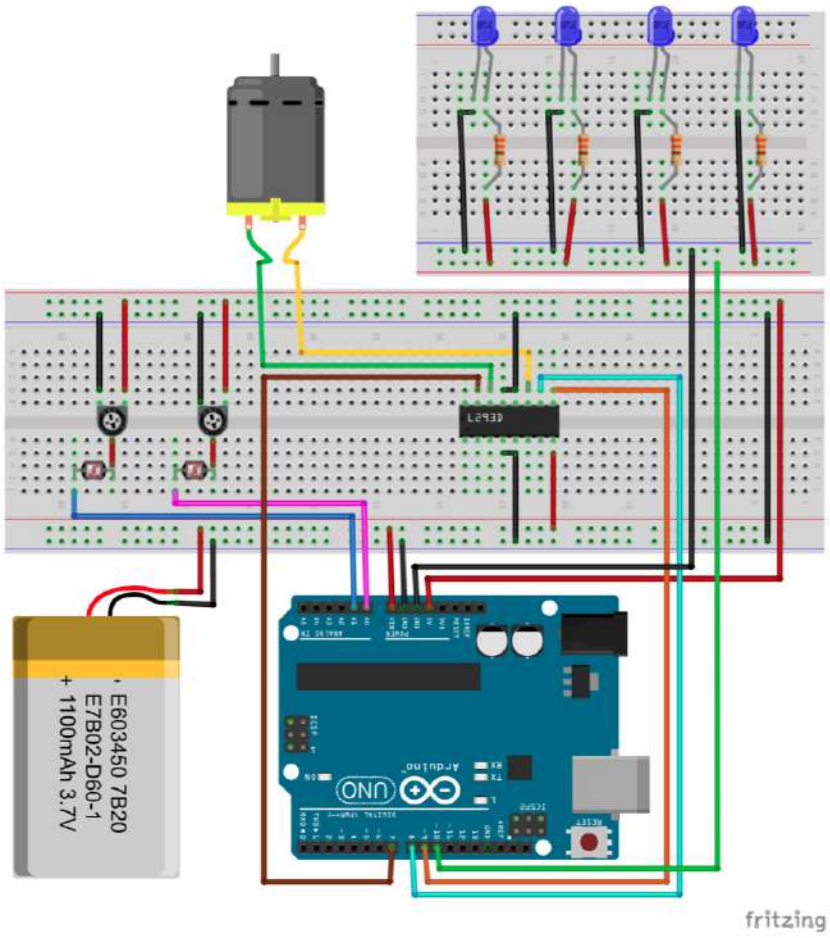


Figura 7.8: Diagrama de conexiones electrónicas del sistema robótico.

El diagrama anterior fue desarrollado buscando representar fielmente las conexiones electrónicas que componen a todo el sistema, por lo que a continuación se presenta el detalle de las conexiones realizadas:

- En la parte superior derecha del diagrama se encuentra una serie de LEDs azules, representando las luces que proveen de iluminación externa al sistema. Cada una de estas luces requiere de una resistencia con un valor de 330Ω (330 Ohms), por lo que en el diagrama también se incluyen estas resistencias.

- Los sensores de iluminación se conforman por una fotoresistencia y un potenciómetro montados sobre una placa, por lo que se tomó la decisión de representarlos con estos mismos elementos dentro del diagrama. El propósito de los potenciómetros es modificar la sensibilidad de las fotorresistencias para decidir qué valores se envían al microcontrolador.

- La bateria del robot se representó con una batería de 3.7v ya que fue el único elemento disponible dentro del software usado, sin embargo fue incluída para ejemplificar el suministro de energía que reciben los componentes elegidos.

- El controlador L293D se representó como un circuito integrado con capacidad para dos motores, y fue ubicado al centro del diagrama. Como puede observarse, el motorreductor se encuentra conectado a uno de los puertos del controlador, ejemplificando la conexión realizada entre el mismo motorreductor y la placa usada en este proyecto.

- Las conexiones entre los diferentes elementos electrónicos y el microcontrolador fueron las siguientes:

    - Se energizó el Arduino con la pila de 12v mediante el **pin Vin**, el cual es un pin de entrada con el propósito específico de energizar el microcontrolador tolerando voltajes de 6v a 12v.
    - Para energizar los sensores de iluminación se usó el **pin 5v**, el cual proporcionó la corriente necesaria a cada uno de los potenciómetros, y estos a su vez brindaron la corriente proporcional a la fotorresistencia de acuerdo con la sensibilidad aplicada por el potenciómetro. Los valores enviados por los sensores fueron leídos con ayuda de los **pines A0** y **A1**, los cuales son pines analógicos que fueron configurados como pines de entrada para realizar las lecturas.
    - Las luces LED fueron energizadas de forma paralela con ayuda del **pin 10**, el cual es un pin tipo PWM que fue configurado como un pin de salida con el propósito de variar la intensidad que las luces debieran encender.
    - El controlador L293D también fue energizado con la pila de 12v mediante el **pin VMS**, el cual es un pin de entrada que permite

energizar el controlador tolerando voltajes desde los 6v hasta los 12v. El **pin 7** y el **pin 8** fueron usados como pines de salida, permitiendo indicar la dirección en la que el motorreductor debiera girar, mientras que el **pin 9** fue usado para indicar la habilitación o deshabilitación del motorreductor.

El código desarrollado para la interacción entre el microcontrolador y los componentes electrónicos y periféricos puede ser consultado en el Apéndice B bajo la etiqueta B.1, mientras que el resultado del trabajo realizado durante la etapa de la implementación e integración de los componentes puede visualizarse en la Figura 7.9, la cual muestra al sistema robótico en su etapa final.

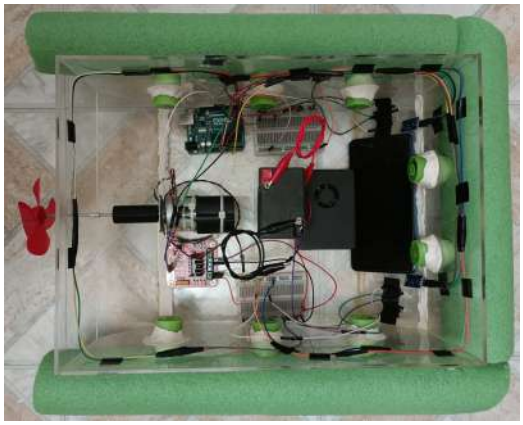


Figura 7.9: Sistema robótico con todos sus elementos integrados.

Capítulo 8

RESULTADOS

Para concluir con la serie de actividades que propone la metodología en cascada, la etapa final en el diseño e implementación de este proyecto consiste en realizar un análisis y evaluación de los resultados obtenidos con la solución propuesta. La intención de esta etapa es poder comparar el comportamiento que tienen el robot y el sistema de detección de objetos con el comportamiento esperado. De esta manera, podrán ser identificadas las áreas de opotunidad para mejorar la solución y se podrá llegar a conclusiones basadas en los experimentos realizados.

A lo largo de este capítulo se presentan los resultados más importantes que fueron obtenidos después de la implementación de la solución. La primera parte de este capítulo se concentra en explicar cómo fueron elegidos los mejores modelos para detección de objetos considerando algunas métricas de evaluación y recursos visuales. Después se muestran algunas evidencias del sistema robótico actuando sobre un ambiente acuático controlado. Por último, se incluyen algunas pruebas empleadas y sus respectivos resultados usando imágenes controladas y no controladas ante los algoritmos de detección de objetos elegidos.

8.1. Sistema de detección de objetos

Como se mencionó en el apartado de *Implementación*, se desarrollaron varios modelos de detección de objetos basados en las arquitecturas de YOLOv3 y YOLOv4, los cuales fueron comparados bajo dos principales características: **el conjunto de imágenes** usado para el proceso de entrenamiento, y **los pesos resultantes** de las neuronas en cada bloque de iteraciones. Para la elección del mejor modelo se requirió de una correcta evaluación del desempeño de cada uno de ellos, por lo que fueron usadas las métricas de evaluación presentadas en el apartado *Comprensión teórica*.

Adicionalmente, los modelos fueron evaluados considerando las **gráficas de pérdida** generadas con ayuda de Darknet [53]. El propósito de estas gráficas es poder visualizar la tasa de errores en los que incurre el modelo conforme se va realizando el proceso de entrenamiento, así como poder encontrar los pesos óptimos de las neuronas para los cuales la función de pérdida de cada arquitectura de YOLO es minimizada sin *sobreajustar el modelo*.

En las siguientes subsecciones se muestran las gráficas de pérdida generadas para cada uno de los modelos entrenados, así como los resultados obtenidos con las métricas de evaluación para los pesos seleccionados de acuerdo con cada conjunto de imágenes de entrenamiento.

8.1.1. Modelos generados con el *Conjunto 1* de imágenes

Como se definió anteriormente, el conjunto 1 de imágenes está conformado por 2040 fotografías de objetos plásticos de las clases de interés. Tomando en cuenta el muestreo aleatorio realizado, el subconjunto de entrenamiento para estos modelos fue de 1897 imágenes, mientras que el subconjunto de validación se conformó de 143 imágenes. Como resultado del análisis de las gráficas de pérdida y las métricas de evaluación sobre todos los pesos generados, se escogieron los pesos correspondientes a las **2000 iteraciones** del proceso de entrenamiento para ambas arquitecturas.

Las gráficas de pérdida asociadas a los modelos se presentan a continuación, donde el eje horizontal de las gráficas muestra un aproximado de las iteraciones

realizadas durante el proceso de entrenamiento, mientras que el eje vertical muestra la pérdida calculada en cada iteración:

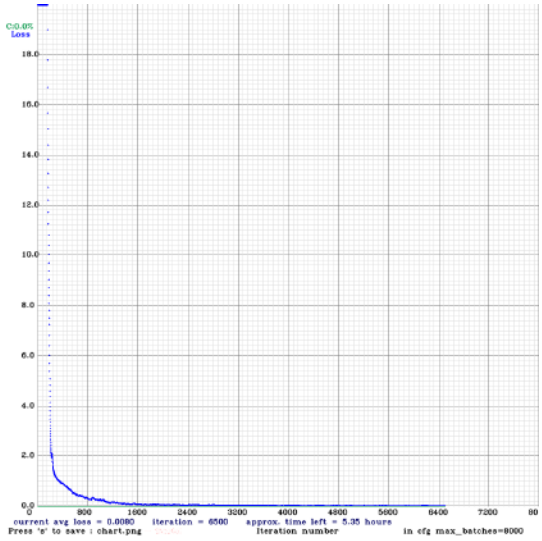


Figura 8.1: Gráfica de pérdida de la arquitectura YOLOv3 con el conjunto 1 de imágenes.

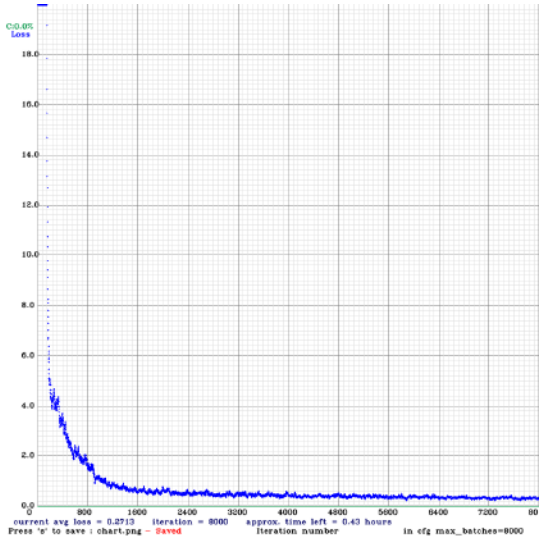


Figura 8.2: Gráfica de pérdida de la arquitectura YOLOv4 con el conjunto 1 de imágenes.

Los resultados de las métricas de evaluación calculadas sobre el subconjunto de validación de imágenes para este modelo se presentan en la tabla 8.1.

Evaluación del desempeño de los modelos					
Arquitectura \Métrica	Precision	Recall	F1-score	mAP	Pesos (iteraciones)
YOLOv3	100 %	100 %	100 %	100 %	2000
YOLOv4	98 %	100 %	99 %	99.82 %	2000

Tabla 8.1: Evaluación de los modelos para los pesos elegidos.

Como puede observarse, el desempeño de ambos modelos parece ser muy bueno ya que los resultados de las métricas que lo evalúan son bastante altos, además de que les toma muy pocas iteraciones minimizar sus respectivas funciones de pérdida. Sin embargo, más adelante se podrá observar su desempeño sobre ejemplos de imágenes nunca antes vistas, lo que nos permitirá juzgar si el modelo está siendo sobrajustado o no.

8.1.2. Modelos generados con el *Conjunto 2* de imágenes

El siguiente conjunto de imágenes es el conjunto 2, el cual está conformado por 2040 fotografías de objetos plásticos de las clases de interés y 4080 imágenes artificiales, dando un total de 6120 imágenes. Tomando en cuenta el muestreo aleatorio realizado, el subconjunto de entrenamiento para estos modelos fue de 5192 imágenes, mientras que el subconjunto de validación fue de 428 imágenes. Como resultado del análisis de las gráficas de pérdida y las métricas de evaluación sobre todos los pesos generados, los pesos escogidos para la arquitectura YOLOv3 fueron los pesos correspondientes a las **5000 iteraciones** del proceso de entrenamiento, mientras que para la arquitectura YOLOv4 se escogieron los pesos correspondientes a las **3000 iteraciones**.

Las gráficas de pérdida asociadas a los modelos se presentan a continuación, donde el eje horizontal de las gráficas muestra un aproximado de las iteraciones realizadas durante el proceso de entrenamiento, mientras que el eje vertical muestra la pérdida calculada en cada iteración:

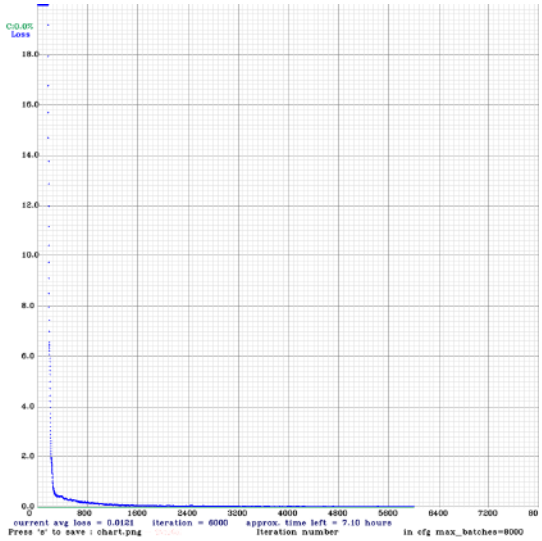


Figura 8.3: Gráfica de pérdida de la arquitectura YOLOv3 con el conjunto 2 de imágenes.

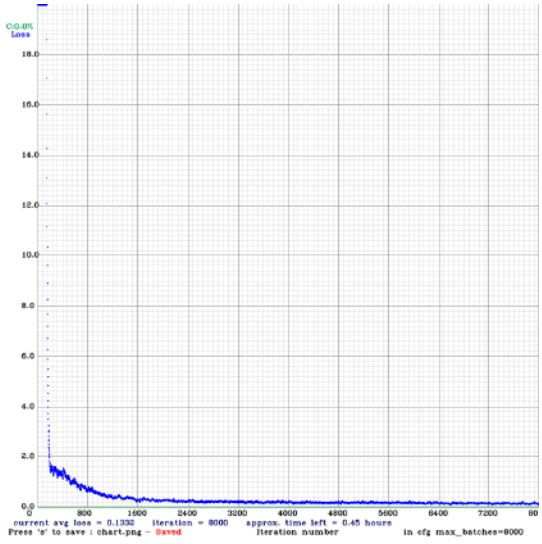


Figura 8.4: Gráfica de pérdida de la arquitectura YOLOv4 con el conjunto 2 de imágenes.

Los resultados de las métricas de evaluación calculadas sobre el subconjunto de validación de imágenes para este modelo se presentan en la tabla 8.2.

Evaluación del desempeño de los modelos					
Arquitectura \Métrica	Precision	Recall	F1-score	mAP	Pesos (iteraciones)
YOLOv3	91 %	98 %	94 %	96.31 %	5000
YOLOv4	100 %	100 %	100 %	100 %	3000

Tabla 8.2: Evaluación de los modelos para los pesos elegidos.

Como puede observarse, el desempeño de ambos modelos difiere un poco a pesar de usar el mismo conjunto de imágenes para el proceso de entrenamiento, por lo que se puede usar como ejemplo para mostrar el impacto que tienen las distintas arquitecturas sobre la detección de objetos. También se puede notar que los pesos elegidos para cada modelo no corresponden a la misma cantidad de iteraciones, por lo que el tiempo que toma minimizar ambas funciones de pérdida pareciera verse impactado conforme se aumenta la cantidad de imágenes según la arquitectura usada.

8.1.3. Modelos generados con el *Conjunto 3* de imágenes

El último conjunto de imágenes es el conjunto 3, el cual está conformado por 2040 fotografías de objetos plásticos de las clases de interés, 2040 imágenes adicionales, y 34722 imágenes artificiales, teniendo un total de 38802 imágenes. Tomando en cuenta el muestreo aleatorio realizado, el subconjunto de entrenamiento para estos modelos fue de 36806 imágenes, mientras que el subconjunto de validación fue de 2716 imágenes. Como resultado del análisis de las gráficas de pérdida y las métricas de evaluación sobre todos los pesos generados, los pesos escogidos para la arquitectura YOLOv3 fueron los correspondientes a las **7000 iteraciones** del proceso de entrenamiento, mientras que para la arquitectura YOLOv4 se escogieron los pesos correspondientes a las **5000 iteraciones**.

Las gráficas de pérdida asociadas a los modelos se presentan a continuación, donde el eje horizontal de las gráficas muestra un aproximado de las iteraciones

realizadas durante el proceso de entrenamiento, mientras que el eje vertical muestra la pérdida calculada en cada iteración:

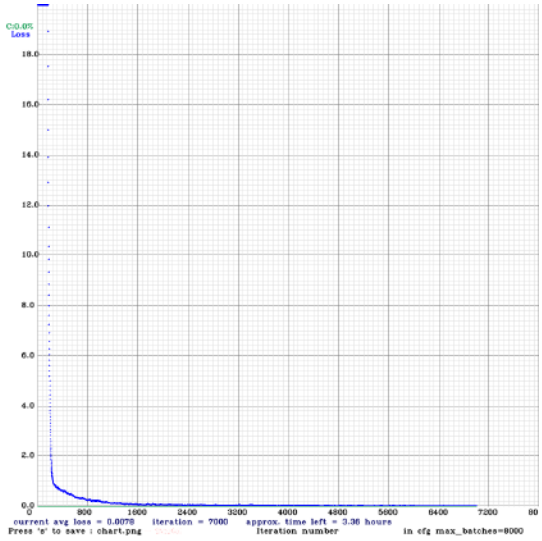


Figura 8.5: Gráfica de pérdida de la arquitectura YOLOv3 con el conjunto 3 de imágenes.

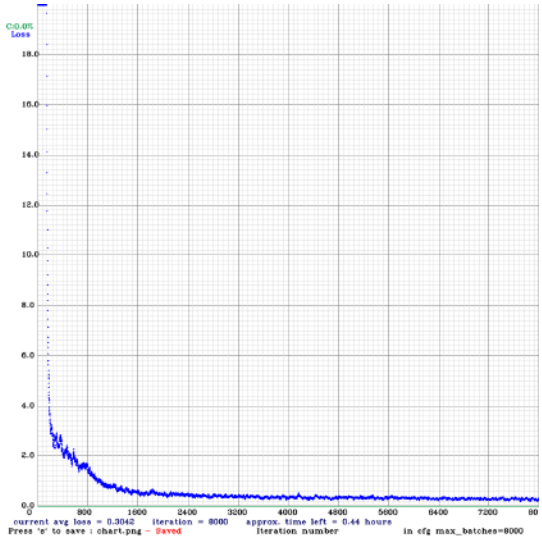


Figura 8.6: Gráfica de pérdida de la arquitectura YOLOv4 con el conjunto 3 de imágenes.

Los resultados de las métricas de evaluación calculadas sobre el subconjunto de validación de imágenes para este modelo se presentan en la tabla 8.3

Evaluación del desempeño de los modelos					
Arquitectura \Métrica	Precision	Recall	F1-score	mAP	Pesos (iteraciones)
YOLOv3	99 %	100 %	100 %	99.99 %	7000
YOLOv4	100 %	100 %	100 %	99.94 %	5000

Tabla 8.3: Evaluación de los modelos para los pesos elegidos.

En este caso, el desempeño de ambos modelos vuelve a ser muy similar para los pesos elegidos, sin embargo podemos notar una diferencia en la cantidad de iteraciones que tomó generar dichos pesos. A partir de estos resultados, se puede observar cómo el tiempo de entrenamiento que toma en generar los pesos ideales efectivamente se ve impactado por la cantidad de imágenes usadas, además de la composición de la arquitectura de la red neuronal elegida.

8.2. Sistema robótico

Continuando con la descripción de los resultados, el funcionamiento del sistema robótico fue evaluado antes de integrar el algoritmo de detección de objetos a la microcomputadora. La intención de hacer esta evaluación por separado fue la de verificar que la estructura del robot cumplía con las expectativas y objetivos definidos, así como asegurar que el comportamiento de navegación e iluminación eran los adecuados según el diseño propuesto.

Para llevar a cabo esta evaluación, el robot fue probado sobre un ambiente acuático controlado, el cual fue recreado dentro de un recipiente plástico con el tamaño suficiente para poder contener agua y al mismo robot. El agua vertida en el contenedor fue mezclada con azúcares, tierra y colorante negro, buscando simular el ambiente acuático objetivo y cumplir con los requerimientos planteados en el apartado *Análisis del problema*.

Las evidencias presentadas a continuación muestran el funcionamiento exitoso del sistema robótico en el ambiente acuático, cumpliendo con el requerimiento de flotación sobre la superficie acuática, así como la navegación de la estructura

con los componentes electrónicos y periféricos contenidos en ella. La Figura 8.7 muestra al sistema robótico dentro del contenedor de plástico y el agua simulada con alta iluminación ambiental, mientras que la Figura 8.8 muestra al sistema robótico dentro del mismo contenedor y el agua simulada pero sin iluminación ambiental, por lo que el robot tuvo que aumentar la intensidad de iluminación con las luces LED para brindar la mayor visibilidad posible a la cámara.

Figura 8.7: Sistema robótico en ambiente acuático con iluminación ambiental.

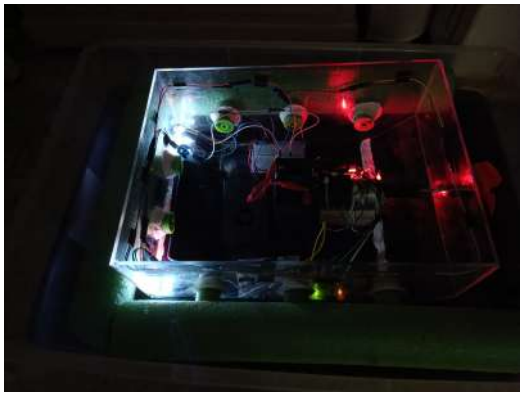


Figura 8.8: Sistema robótico en ambiente acuático sin iluminación ambiental.

8.3. Pruebas individules de los modelos de detección objetos, elección del mejor, e integración con el robot

La tarea final en el proceso de obtención de resultados fue la integración del sistema de detección de objetos en la microcomputadora del robot. De acuerdo con lo detallado en el apartado *Diseño de la solución*, el propósito de usar las arquitecturas de YOLO para generar modelos de detección es realizar las tareas de identificación y clasificación en tiempo real usando una cámara y aprovechando el procesamiento de la microcomputadora, por lo que resulta factible analizar el desempeño de la solución bajo ciertas condiciones.

Inicialmente se tomó la decisión de poner a prueba los modelos de detección usando imágenes similares a las imágenes usadas durante el proceso de entrenamiento. La intención de realizar estas pruebas es poder identificar si los modelos con los pesos elegidos tienen un desempeño cercano al deseado, o en su defecto identificar si debieran hacerse mejoras o seleccionar pesos correspondientes a otros bloques de iteraciones. Para esta tarea se usaron 15 imágenes de prueba provenientes del dataset Wadaba con objetos de las clases de interés, las cuales no fueron usadas durante la fase de entrenamiento de los modelos.

El resultado de todos los modelos con los pesos elegidos fue el esperado: se realizó la detección y clasificación de los objetos plásticos para las clases de interés correctamente. Las Figuras 8.9, 8.10 y 8.11 son una muestra de los resultados obtenidos[1] con los modelos desarrollados, en las cuales pueden observarse diferentes objetos plásticos y su detección realizada exitosamente.

[1]Los resultados fueron obtenidos con un código en Python basado en el código proveído por Valentyn Sichkar en su Github personal: https://github.com/sichkar-valentyn

Figura 8.9: Objeto tipo PET detectado correctamente con la arquitectura YOLOv3 y pesos de 2000 iteraciones.

Figura 8.10: Objeto tipo PEHD detectado correctamente con la arquitectura YOLOv4 y pesos de 3000 iteraciones.

Figura 8.11: Objeto tipo PET detectado correctamente con la arquitectura YOLOv4 y pesos de 5000 iteraciones.

Algo que puede destacarse de los resultados anteriores es que en la Figura 8.9, que fue resultado del modelo generado con el conjunto 1 de imágenes, la detección del objeto se realizó con dos distintos bounding boxes, sin embargo, ninguno de ellos se ajusta correctamente al tamaño del objeto a pesar de que la clasificación del tipo de plástico y su posición es correcta. En cambio, en las Figuras 8.10 y 8.11, que fueron resultado de los modelos generados con los conjuntos 2 y 3 de imágenes respectivamente, se puede observar que los bounding boxes asociados a la detección de cada objeto fueron ajustados correctamente según el tamaño del mismo. Con estos resultados puede inferirse que involucrar imágenes adicionales en el proceso de entrenamiento, así como la elección de pesos asociados a iteraciones mayores, puede repercutir en la calidad de las detecciones.

Una evaluación adicional en el desempeño de los modelos fue usar imágenes que no fueran capturadas con un fondo negro o con poca iluminación debajo del agua, es decir, imágenes que no cumplieran con el estándar del problema a resolver. Para este caso se usaron 10 imágenes con distintos objetos plásticos y colores de fondos, tratando de verificar si el modelo podía identificar alguno de ellos o realizaba algún tipo de detección por color a pesar de no ajustarse a las restricciones del problema. Como era de esperarse, los objetos no fueron detectados con ninguna de las configuraciones de arquitecturas y pesos, lo que nos permite deducir que el modelo se está ajustando correctamente al problema. Las Figuras 8.12 y 8.13 son ejemplos de imágenes con objetos plásticos de las clases de interés en las que no se realizó alguna detección.

Figura 8.12: Objeto de tipo PS no identificado.

Figura 8.13: Objeto de tipo PEHD no identificado.

La última fase de pruebas involucró la integración del sistema de detección de objetos con la microcromputadora y la estructura del sistema robótico para evaluar si el comportamiento obtenido con la solución completa era el resultado esperado. Para esta fase se tuvieron que considerar algunos aspectos adicionales como la calidad de las fotografías, el tiempo de procesamiento de cada imagen por la microcomputadora, el tiempo entre capturas de fotografías, y la iluminación con la que contaba el ambiente.

Después de una serie de comparaciones entre tiempos de procesamiento y calidad de las imágenes capturadas, se optó por usar la cámara del celular como dispositivo para tomar las fotografías del ambiente navegado. La razón de usar este elemento dentro del sistema completo es que tiene un menor tiempo de retardo entre capturas de imágenes, además de que la calidad de cada fotografía es mayor en comparación con las capturadas con la cámara GoPro. Teniendo en cuenta esta decisión, los resultados que se presentan a continuación fueron generados a partir de fotografías de objetos plásticos sumergidos en el ambiente acuático controlado y que fueron capturados con la cámara elegida.

También se determinó que los modelos que producían los mejores resultados fueron los modelos generados a partir del conjunto 2 de imágenes, sin embargo fue necesario elegir pesos asociados a bloques de iteraciones posteriores ya que el desempeño de ambos modelos mejoró considerablemente, por lo que los pesos finales fueron los asociados a las 6000 iteraciones para cada uno de ellos. Las siguientes figuras son ejemplos comparables de los resultados obtenidos con las arquitecturas y pesos finales después de haberse integrado a la solución.

Figura 8.14: Objeto detectado con la arquitectura YOLOv3.

Figura 8.15: Objeto detectado con la arquitectura YOLOv4.

Las Figuras 8.14 y 8.15 muestran la identificación de un objeto de clase PS realizado exitosamente con cada uno de los modelos finales. En la Figura 8.14 puede observarse que el bounding box de la detección está siendo ajustado con menor precisión ya que está considerando información contextual de la imagen como parte del objeto, mientras que en la Figura 8.15 se puede observar que el bounding box se ajustó correctamente al tamaño del objeto sin considerar parte del contexto ambiental.

Respecto a la clasificación del objeto, ambos modelos incurrieron en un error al clasificarlo como objeto tipo PEHD en vez de tipo PS, lo que permite inferir que posiblemente este tipo de errores se deben a que el objeto no fue capturado por completo, o en su defecto que se necesitan más imágenes de las clases menos representadas para la etapa de entrenamiento, lo que permitiría tener mayor generalización para todas las clases a la hora de realizar alguna clasificación.

Figura 8.16: Objeto detectado con la arquitectura YOLOv3.

Figura 8.17: Objeto detectado con la arquitectura YOLOv4.

En el caso de las Figuras 8.16 y 8.17 se puede observar que ambos modelos arrojan resultados muy similares, identificando correctamente la posición del objeto y considerando aproximadamente regiones similares para predecir el bounding box correspondiente. En la Figura 8.16 puede observarse que el modelo detecta parte de la botella de plástico y se enfoca en la etiqueta de publicidad de la misma, mientras que en la Figura 8.17 la identificación del objeto se realiza considerando parte de la deformación de la botella y un fragmento inferior de la misma, generando de esta manera el bounding box para una mayor proporción del objeto.

Respecto a la clasificación del objeto, ambos modelos lo asociaron correctamente al tipo de plástico PET, lo cual es consistente con el resultado esperado para este tipo de objetos. Sin embargo, es necesario tomar en cuenta que la clase PET es la clase más representada en los subconjuntos de imágenes de

entrenamiento, por lo que tendríamos que hacer una tarea de balanceo de clases más robusto para probar los modelos y corroborar que no están sesgados.

Figura 8.18: Objeto no detectado con la arquitectura YOLOv3.

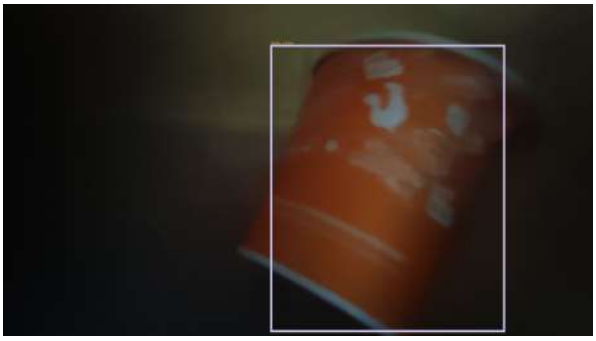


Figura 8.19: Objeto detectado con la arquitectura YOLOv4.

El último caso a comparar es el de las Figuras 8.18 y 8.19, donde el desempeño de la detección del objeto es distinto entre modelos. Antes de explicar la evaluación es importante destacar que la imagen usada para la detección carece de la resolución deseada, además de que fue capturada mientras el robot se encontraba en movimiento, por lo que el objeto pareciera estar difuminado o con un poco de ruido.

En cuanto al desempeño de los modelos, la Figura 8.18 muestra que el modelo creado con la arquitectura YOLOv3 no logró realizar alguna detección, por lo que podemos decir que el modelo es poco tolerante al ruido o a variaciones provocadas por el movimiento del robot. En cambio, en la Figura 8.19 puede

observarse que el modelo identificó correctamente la posición del objeto y ajustó el bounding box al tamaño del mismo. Desafortunadamente la clasificación del objeto fue errónea ya que fue asociado a la clase PEHD en vez de la clase PS, por lo que podemos reforzar la teoría de que, al ser clases poco representadas, es altamente probable que el modelo no logre a generalizar la información necesaria para discernir entre ambas clases (PS y PEHD), por lo que aumentar el tamaño de muestra de ambas clases para la etapa de entrenamiento sería una solución que podría incrementar el desempeño en la clasificación.

Como nota adicional, para obtener los resultados anteriormente descritos se usó un código en Python basado en el código proveído por Valentyn Sichkar [62] y adaptado a la necesidad de implementar detección en tiempo real con ayuda de la cámara del celular. Este código puede ser consultado en el Apéndice A bajo la etiqueta A.4.

8.4. Resumen de los resultados

Con la intención de concentrar los resultados obtenidos después de la implementación de la solución, la Tabla 8.4 y la Figura 8.20 muestran un resumen del desempeño del modelo seleccionado considerando las métricas de evaluación presentadas y los subconjuntos de imágenes de validación y prueba.

Resumen del desempeño final del modelo					
Subconjunto \Métrica	Tamaño	Objetos plásticos	Precision	Recall	F1-score
Validación	428	213	100	100 %	100 %
Prueba (sin agua)	15	15	100 %	100 %	100 %
Prueba (con agua)	15	15	51.85 %	53.33 %	48.80 %

Tabla 8.4: Métricas de evaluación para el desempeño del modelo final.

En la tabla anterior pueden observarse dos subconjutos de prueba, los cuales corresponden a las imágenes provenientes del dataset WaDaBa que no fueron usadas durante la etapa de entrenamiento ni validación, y las imágenes de los objetos que fueron capturados en el ambiente acuático simulado. Se decidió calcular las métricas de evaluación sobre ambos subconjuntos de imágenes para

poder contrastar los resultados obtenidos y evaluar si el desempeño obtenido corresponde el desempeño deseado para nuestra solución. En la columna **Tamaño** se reporta la cantidad de imágenes que contenía cada subconjunto, la columna **Objetos plásticos** reporta la cantidad de imágenes con objetos plásticos a ser detectados, mientras que las columas **Precision, Recall y F1-score** resportan el promedio de las métricas individuales aplicadas a cada clase de interés.

Para este caso no se está considerando la métrica ***Intersection Over Union*** que calculan los modelos de YOLO durante su proceso de entrenamiento [54]. La razón de no incluirla es que esta métrica está relacionada con el etiquetado sobre las imágenes de entrenamiento y el bounding box predicho, por lo que en caso de un mal etiquetado esta métrica podría verse afecta, derivando en un resultado subjetivo de acuerdo con la metodología seguida para el etiquetado.

Una representación visual del desempeño del modelo sobre el subconjunto de imagénes de prueba capturadas en el ambiente acuático puede observarse en la Figura 8.20, la cual es una *matriz de confusión multiclase* generada con la librería MLXTEND de Python que muestra los errores y aciertos en los que incurrió el modelo de acuerdo con nuestras clases objetivo y las clases predichas. Esta matriz de confusión puede interpretarse de la siguiente manera:

- Se usaron 15 imágenes de prueba en el ambiente acuático: 5 con objetos de clase PET, 5 con objetos de clase PS, y 5 con objetos de clase PEHD.
- Para la clase PET, todos los objetos de la clase fueron detectados correctamente.
- Para la clase PS, 1 objeto de la clase fue detectado correctamente, mientras que 3 objetos fueron asociados a la clase PET y 1 objeto fue asociado a la clase PEHD.
- Para la clase PEHD, 2 objetos de la clase fueron detectados correctamente, mientras que 2 objetos fueron asociados a la clase PS y 1 objeto fue asociado a la clase PET.

En conjunto con la interpretación anterior, la matriz de confusión multiclase permitió corroborar los cálculos de las métricas de evaluación presentadas en la Tabla 8.4, por lo que resulta una buena alternativa para entender el comportamiento del modelo ante imágenes que no fueron parte de los subconjuntos de entrenamiento o validación.

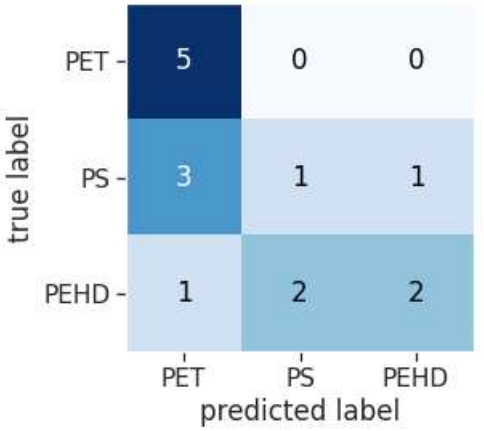


Figura 8.20: Matriz de confusión para el conjunto de imágenes de prueba capturadas en un ambiente acuático.

Capítulo 9

CONCLUSIONES

Para finalizar con el diseño e implementación de la solucion propuesta, en este capítulo se incluye una serie de conclusiones que fueron generadas durante el desarrollo de la solución desarrollada. El propósito de este capítulo es discutir algunas observaciones que surgieron a partir de la solución hasta su implementación y evaluación, además de plantear una línea futura de trabajo que permita añadir mejoras a la solución desarrollada y dar una solución más precisa al problema planteado.

9.1. Análisis del sistema de detección de objetos

Uno de los más grandes aciertos dentro del desarrollo de la solución fue el sistema de detección de objetos, el cual se desempeña de forma exitosa y cumple con las necesidades y restricciones planteadas en la definición del problema. Considerando que uno de los alcances principales de la solución era generar un modelo que funcionara bajo un ambiente controlado simulando una situación real, los modelos elegidos lograron generalizar el problema a resolver con un conjunto de datos limitado, por lo que podemos concluir que las actividades realizadas sobre los conjuntos de imágenes, desde su reescalamiento hasta la

generación de imágenes artificiales, además de la combinación de configuraciones para cada arquitectura, fueron elecciones que beneficiaron el desempeño del modelo.

De manera independiente, los modelos elegidos se desempeñaron correctamente de acuerdo con las métricas de evaluación consideradas. Ambos modelos identificaron a los objetos plásticos presentes en las imágenes capturadas fuera del agua, además de que la clasificación de cada uno de ellos se realizó exitosamente a pesar de contar con un dataset desbalanceado en cuanto a la cantidad de imágenes por clase de plástico considerada. A partir de este análisis se puede inferir que si se tuviera un dataset balanceado se podría generalizar aún más el problema de clasificación, esperando mejores resultados y reduciendo la proporción de errores.

9.2. Análisis del sistema robótico

Otro gran acierto en el desarrollo de la solución fue el modelado e implementación de una estructura sólida y funcional del sistema robótico, la cuál fue diseñada para contener a todos los elementos electrónicos y periféricos involucrados en el diseño de la solución, además de que cumpliera con todos los objetivos y restricciones que se plantearon. Considerando que el diseño de la estructura desarrollada se basó en el diseño de proyectos previamente elaborados y publicados, el diseño implementando involucró una serie de mejoras para disminuir el peso del sistema, así como facilitar su movilidad sobre la superficie acuática y permitir que fuera escalable en cuanto a componentes y movilidad.

En general el diseño de la estructura permitió distribuir cada uno de los componentes elegidos de forma correcta, permitiendo una reparación o sustitución de los componentes de forma sencilla sin perder de vista el diseño final. Así mismo, el sistema de flotación en conjunto con el sistema de propulsión permiten que pueda añadirse más peso dentro de la estructura sin afectar su desempeño, lo que corrobora la idea de ser una estructura escalable y funcional.

9.3. Análisis de la solución integrada

Considerando los resultados obtenidos durante la etapa de pruebas, se puede asegurar que el problema planteado fue resuelto casi por completo considerando las restricciones y el alcance definido. En general, la estructura del sistema robótico pudo navegar sobre los ambientes acuáticos simulados sin problema, por lo que en conjunto con el sistema de detección de objetos se lograron realizar las tareas de exploración y detección de objetos de manera exitosa.

Por otro lado, no se logró cumplir con el requerimiento de otorgar resultados sobre la detección de objetos en tiempo real debido a la elección de la microcomputadora en conjunto con las arquitecturas YOLO. Al poner a prueba el procesamiento de la Raspberry Pi con los modelos seleccionados, se tuvo un tiempo promedio de procesamiento de 2 segundos por cada imagen, lo que nos permite asegurar que la capacidad computacional es limitada. Aunado a esta prueba, en el reporte técnico que describe el funcionamiento de YOLO en su versión 1 se menciona que se pueden obtener resultados en tiempo real usando una unidad de procesamiento gráfico adicional [54], por lo que al usar una microcomputadora como la Raspberry Pi no se cuenta con el suficiente poder computacional para ejecutar todas las operaciones involucradas en cada arquitectura velozmente.

A pesar de no cumplir con el requerimiento anteriormente descrito, el desempeño de la solución en cuanto a la clasificación de los objetos debajo del agua fue satisfactoria. Cada uno de los objetos capturados fue identificado sin importar su posición o combinación de colores, aún cuando el ambiente simulado no contaba con la suficiente iluminación para la cámara fotográfica. Respecto a la clasificación de los objetos, se puede notar un sesgo hacia la clase mayormente representada en el conjunto de imágenes, que es la clase PET, por lo que se puede inferir que se requiere balancear de mejor forma el dataset para reducir los errores de clasificación y generalizar de mejor forma el problema.

9.4. Trabajo futuro

Teniendo en mente la mejora de la solución desarrollada, se tiene contemplada la recolección de más imágenes de objetos plásticos en ambientes acuáticos y ambientes controlados, lo que permitiría generalizar con mayor éxito al problema planteado y brindaría mayor robustez a la hora de clasificar los objetos detectados. Como consecuencia, también se tiene considerado balancear la cantidad de imágénes totales por clase con la intención de reducir el sesgo a la hora de clasificar un objeto detectado.

En cuanto a la asignación de tareas para cada uno de los módulos que involucran a la solución, el uso de una microcomputadora Raspberry Pi para el procesamiento de imágenes y ejecución del algoritmo de detección de objetos resulta ser computacionalmente insuficiente, por lo que se debe realizar una sustitución de este componente por una computadora externa que realice ambas tareas y facilite cumplir con el requerimiento de análisis en tiempo real. Así mismo, se debe considerar la implementación de múltiples configuraciones de arquitecturas YOLO con diversas combinaciones de hiperparámetros, por lo que realizar una migración de los procesos de entrenamiento y validación de los modelos a plataformas que faciliten su replicación brindaría mayor facilidad para encontrar el mejor modelo.

También es necesario considerar arquitecturas alternativas de redes neuronales que se ajusten con los requerimientos planteados para este proyecto. Usar arquitecturas más complejas resultaría en mayor tiempo para proporcionar resultados, mientras que al usar arquitecturas reducidas posiblemente no se obtendrían los resultados deseados. De esta manera, es necesario hacer una investigación sobre cuáles son las arquitecturas que mejor se ajustan a los requerimientos y restricciones definidos, así como llevar a cabo su implementación y evaluación con la información disponible.

En cuanto a la estructura del sistema robótico, una buena alternativa a la navegación del sistema puede ser el brindar movimiento por radiofrecuencia, lo que facilitaría la exploración de zonas específicas en caso de ser necesario. Para cumplir con la mejora anterior, se debe modificar el diseño de la estructura para incluir un par de motores que brinden dirección, así como adaptar el algorit-

mo de movimiento para que puedan interactuar con el motor que propulsa la estructura.

Respecto a la cámara fotográfica, se pueden explorar alternativas de cámaras con capacidad de ser sumergidas bajo el agua y que mejoren la resolución de las fotografías capturadas, beneficiando la visibilidad de la cámara y la detección de los objetos más profundos. En conjunto con este componente, brindar una mejor distribución de la iluminación artificial facilitaría la captura de fotografías con mayor visibilidad de los objetos, además de proporcionar una iluminación uniforme para toda la estructura.

Por último, se tiene considerado la implementación de la solución en ambientes acuáticos no controlados para verificar que su comportamiento es similar al que tiene en ambientes controlados. Con esto en mente, se requiere otorgar una protección adicional a los componentes electrónicos y periféricos ante cualquier siniestro, por lo que agregar una cubierta a la estructura del sistema robótico que permita una fácil manipulación de dichos componentes brindaría más seguridad y no impediría realizar mejoras o cambios sobre el mismo prototipo.

Capítulo 10

APORTACIONES ADICIONALES

10.1. Aportaciones a la generación de datos artificiales

En el apartado *Recolección y preparación de las imágenes* se detalló el proceso para conseguir y preparar la información necesaria para entrenar el modelo de detección de objetos. También se habló de las técnicas y operaciones usadas para incrementar la cantidad de imágenes usando la herramienta AugLy, así como los respectivos etiquetados que las imágenes requerían. Los aportes que se explican a continuación están relacionados con las técnicas de generación de datos artificiales o data augmentation, donde se solucionó un error de código que tenía la herramienta AugLy, y se generó un recurso de acceso libre para generar imágenes usando esta misma herramienta.

AugLy se caracteriza por ser una aplicación relativamente nueva con poco menos de un año y medio desde su fecha de lanzamiento (17 de junio del 2021) [43], además de ser una herramienta de *software libre* que permite las aportaciones de la comunidad con miras a mejorar sus implementaciones. Debido a su lanzamiento reciente, AugLy sigue en constante desarrollo ya que requiere

correcciones de errores de código o mejoras para las operaciones que realiza.

Durante la aplicación de las técnicas de data augmentation a las imágenes recolectadas, se estableció la tarea de revisar los procedimientos y operaciones que se realizaron con la intención de entender su funcionamiento. Durante esta tarea se encontró que la técnica *Color jitter* contaba con un error a la hora de realizar las alteraciones de las propiedades de colores de las imágenes, lo que impedía obtener el resultado deseado ante ciertas instrucciones.

La función recibe como parámetros los valores asociados a las propiedades de la imagen, los cuales definen la intensidad de alteración que recibe cada propiedad. Sin embargo, si una o más de las propiedades no querían ser modificadas, y únicamente se proporcionaban los valores para las propiedades que se requería modificar, la imagen no recibía alteración en ninguna de sus propiedades. El error se notificó mediante la plataforma Github emitiendo una *petición para corrección de errores*[1], donde se explicó el error encontrado y se mostraron ejemplos que sustentaban la petición. El error fue revisado por una de las ingenieras que desarrollan la aplicación, aprobando el cambio y realizando las correcciones pertinentes para solucionarlo.

Después de notificar el error encontrado y realizar pruebas de las funciones que contiene AugLy, se estableció la tarea de generar un recurso libre que sirviera de consulta para desarrollar otros proyectos que requieran data augmentation con imágenes. Este recurso fue desarrollado usando un *Jupyter Notebook*, permitiendo la ejecución de código en Python por bloques y facilitando la organización y estructura de las operaciones implementadas.

El Jupyter Notebook está compuesto por varias secciones que abarcan la organización de las imágenes, la copia de sus archivos de texto para etiquetado, y la implementación de técnicas individuales y combinadas para data augmentation. Una de las ventajas de este recurso es que aprovecha la compatibilidad de *Google* para ejecutar Jupyter Notebooks, además de usar *Google Drive* para trabajar con los archivos, lo que permite almacenar las imágenes resultantes en el mismo espacio de almacenamiento personal.

[1] Link para consultar la petición: https://github.com/facebookresearch/AugLy/issues/102

Este recurso puede ser consultado y descargado desde la plataforma Github bajo el perfil *BrandonHT*[2], el cual es un perfil personal del autor de este proyecto. El ambiente recomendado para usar el recurso es *Google Colab*, ya que es una plataforma que permite ejecutar Jupyter Notebooks de forma remota aprovechando las compatibilidades de Google anteriormente mencionadas. Dentro del archivo, se pueden modificar las ubicaciones de las imágenes a trabajar y las carpetas destino de los resultados, además de poder cambiar los valores de los parámetros de cada técnica con la intención de obtener distintos resultados.

10.2. Aportaciones al entrenamiento de modelos usando Darknet

Con la intención de proveer una alternativa para el entrenamiento de modelos de detección de objetos a las personas que no tengan acceso a una computadora con tarjeta GPU, se tomó la decisión de desarrollar un recurso de acceso libre que aprovechara algunos elementos de *Google Cloud* y que fuera de fácil replicabilidad. Google Cloud se caracteriza por proveer de software y hardware gratuitos para el desarrollo de proyectos con el objetivo de dar a conocer sus productos, por lo que el recurso desarrollado aprovecha esta ventaja para combinar el uso de una tarjeta GPU gratuita con el desarrollo de un Jupyter Notebook, además de usar el almacenamiento de Google Drive para facilitar la preparación de las imágenes para el entrenamiento de algún modelo.

Gracias a que Google Colab provee un ambiente dinámico para el desarrollo de proyectos en Python, no se requiere de alguna configuración especial para el uso de tarjetas GPU en esta plataforma. Sin embargo, es necesario tener en cuenta que Google Cloud proporciona distintos modelos de tarjetas GPU gratuitas dependiendo de la región del usuario y del uso previo que haya tenido, por lo que el tiempo de entrenamiento de algún modelo puede verse impactado según sea el modelo de tarjeta GPU asignada[3].

[2]Link de Github para consultar el recurso desarrollado con AugLy: https://github.com/BrandonHT/image-augmentation-with-AugLy

[3]Información sobre las tarjetas GPU de Google Cloud: https://cloud.google.com/gpu

El recurso desarrollado consta de dos Jupyter Notebooks que permiten el entrenamiento de modelos de detección de objetos usando las arquitecturas YOLOv3 y YOLOv4 con el framework Darknet. En cada Jupyter Notebook se incluyen las instrucciones detalladas de los procedimientos que el usuario debe realizar, desde el almacenamiento de las imágenes en una ubicación en Google Drive, hasta funciones que permiten preparar correctamente las imágenes para entrenar el modelo. Así mismo, permite guardar los pesos generados en cada etapa de entrenamiento del modelo en una ubicación de Google Drive, por lo que todo el proceso está optimizado para que pueda ser usado por cualquier usuario que posea una cuenta de Google. Al igual que el recurso anterior para generar imágenes usando AugLy, este recurso puede ser consultado y descargado desde la plataforma Github bajo el perfil *BrandonHT*[4]

[4]Link de Github para consultar el recurso desarrollado con Darknet: https://github.com/BrandonHT/YOLOv3-and-YOLOv4-multiclass-object-detector

Apéndice A

CÓDIGOS EN PYTHON USADOS DURANTE EL DESARROLLO

```python
from os import listdir
from os.path import join
import cv2 as cv
import math

source_path = "" # Definir la ubicacion original de las imagenes
dest_path = "" # Definir donde se guardaran las imagenes procesadas
files = [f for f in listdir(source_path)]
# Cada imagen se reescala a un tercio del tamanio original
for img_file in files:
    image_path = join(source_path, img_file)
    image= cv.imread(image_path)
    n, m, c = image.shape
    image = cv.resize(image, (math.ceil(m/3), math.ceil(n/3)))
    cv.imwrite(join(dest_path, img_file), image)
```

CÓDIGO A.1: Código para reescalar las imágenes del dataset WaDaBa.

```python
import cv2 as cv
import math
from os.path import join
import glob


source = "" # Definir la ubicacion original de las imagenes
dest = "" # Definir donde se guardaran las imagenes procesadas
files = glob.glob(f"{source}*.jpg") # filtrar   nicamente   imagenes
dummy_number = 0
# Para cada imagen adicional se reescala por un factor definido,
# se modificar su nombre y se crea un archivo de texto vacio
for img in files:
    dummy_name = f"dummy_image_{dummy_number}"
    dummy_image_name = f"{dummy_name}.jpg"
    dummy_text_name = f"{dummy_name}.txt"
    image = cv.imread(join(source, img))
    h, w, c = image.shape
    if w > h:
        factor = w  / 640
        factor = 1 / factor
    else:
        factor = h / 426
        factor = 1 / factor
    new_image = cv.resize(image, (math.ceil(w*factor), math.ceil(h*
    factor)))
    cv.imwrite(join(dest, dummy_image_name), new_image)
    f = open(join(dest, dummy_text_name), "w+")
    f.close()
    dummy_number += 1
```

CÓDIGO A.2: Código para reescalar las imágenes adicionales.

```python
import glob
from sklearn.model_selection import train_test_split

source = "" # Definir la ubicacion original de las imagenes
images_list = glob.glob(source)

# Crear los subconjuntos de entrenamiento y validacion
train, test = train_test_split(images_list, test_size=0.07)
print(len(train))
print(len(test))

# Agregar las imagenes del conjunto de entrenamiento al registro
# de imagenes de entrenamiento de Darknet
fileTrain = open("darknet/data/train.txt", "w")
fileTrain.write("\n".join(train))
fileTrain.close()

# Agregar las imagenes del conjunto de validacion al registro
# de imagenes de validacion de Darknet
fileTest = open("darknet/data/test.txt", "w")
fileTest.write("\n".join(test))
fileTest.close()
```

CÓDIGO A.3: Código para generar los subconjuntos de imágenes de entrenamiento y validación.

Apéndice A: Códigos en Python usados durante el desarrollo

```
# Algorithm:
# Reading stream video from camera (ip webcam)
# --> Loading essentials for YOLO (names)
# --> Loading YOLO v3 / YOLO v4 Network
# --> Reading frames in the loop
# --> Getting blob from the frame
# --> Implementing Forward Pass
# --> Getting Bounding Boxes
# --> Non-maximum Suppression
# --> Drawing Bounding Boxes with Labels
# --> Showing processed frames in OpenCV Window
#
# Result:
# Window with Detected Objects, Bounding Boxes and Labels in Real Time


# Importing needed libraries
import numpy as np
import cv2
import time
import requests

### Reading stream video from camera

# Defining the URL of the camera to use
# Example of the url could be: http://192.168.0.9:8080/shot.jpg
camera_url = ""

# Preparing variables for spatial dimensions of the frames
h, w = None, None

### Loading essentials for YOLO
# Example of the path could be: /home/ubuntu/Documents/tesis.names
with open('') as f:
    # Getting labels reading every line
    # and putting them into the list
    labels = [line.strip() for line in f]

### Loading YOLO v3/YOLO v4 network
# Loading the network requires the path for network configuration and path for training weights
# Could be used for loading configuration files and weights fotr
```

```
    both yolov4 and yolov4
network = cv2.dnn.readNetFromDarknet("", "")

# Getting list with names of all layers from YOLO network
layers_names_all = network.getLayerNames()

# Getting only output layers' names that we need from YOLO
    algorithms
# with function that returns indexes of layers with unconnected
    outputs
layers_names_output = [layers_names_all[i - 1] for i in network.
    getUnconnectedOutLayers()]

# Setting minimum probability to eliminate weak predictions
probability_minimum = 0.5

# Setting threshold for filtering weak bounding boxes
# with non-maximum suppression
threshold = 0.3

# Generating colours for representing every detected object
# with function randint(low, high=None, size=None, dtype='l')
colours = np.random.randint(0, 255, size=(len(labels), 3), dtype='
    uint8')

### Reading frames in the loop

# Defining loop for catching frames
while True:
    # Capturing frame-by-frame from camera
    img_resp = requests.get(camera_url)
    img_arr = np.array(bytearray(img_resp.content), dtype=np.uint8)
    frame = cv2.imdecode(img_arr, -1)

    # Getting spatial dimensions of the frame
    if w is None or h is None:
        h, w = frame.shape[:2]

    ### Getting blob from current frame

    # The 'cv2.dnn.blobFromImage' function returns 4-dimensional
    blob from current
    # frame after mean subtraction, normalizing, and RB channels
```

```
    swapping
79  # Resulted shape has number of frames, number of channels,
    width and height
80  blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
81                               swapRB=True, crop=False)
82
83  ### Implementing Forward pass to our blob
84
85  network.setInput(blob)  # setting blob as input to the network
86  start = time.time()
87  output_from_network = network.forward(layers_names_output)
88  end = time.time()
89
90  # Showing spent time for single current frame
91  print('Current frame took {:.5f} seconds'.format(end - start))
92
93  ### Getting bounding boxes
94
95  bounding_boxes = []
96  confidences = []
97  class_numbers = []
98
99  # Going through all output layers after feed forward pass
100 for result in output_from_network:
101     # Going through all detections from current output layer
102     for detected_objects in result:
103         # Getting 80 classes' probabilities for current
    detected object
104         scores = detected_objects[5:]
105         # Getting index of the class with the maximum value of
    probability
106         class_current = np.argmax(scores)
107         # Getting value of probability for defined class
108         confidence_current = scores[class_current]
109
110         # Eliminating weak predictions with minimum probability
111         if confidence_current > probability_minimum:
112             # Scaling bounding box coordinates to the initial
    frame size
113             # YOLO data format keeps coordinates for center of
    bounding box
114             # and its current width and height
115             # That is why we can just multiply them elementwise
```

```
116                 # to the width and height of the original frame and
117                 # in this way get coordinates for center
118                 # of bounding box, its width and height for
    original frame
119                 box_current = detected_objects[0:4] * np.array([w,
    h, w, h])
120
121                 # Now, from YOLO data format, we can get top left
    corner coordinates
122                 # that are x_min and y_min
123                 x_center, y_center, box_width, box_height =
    box_current
124                 x_min = int(x_center - (box_width / 2))
125                 y_min = int(y_center - (box_height / 2))
126
127                 # Adding results into prepared lists
128                 bounding_boxes.append([x_min, y_min,
129                                        int(box_width), int(
    box_height)])
130                 confidences.append(float(confidence_current))
131                 class_numbers.append(class_current)
132
133     ### Non-maximum suppression
134
135     # Implementing non-maximum suppression of given bounding boxes
136     # With this technique we exclude some of bounding boxes if
    their
137     # corresponding confidences are low or there is another
138     # bounding box for this region with higher confidence
139
140     # It is needed to make sure that data type of the boxes is 'int
    '
141     # and data type of the confidences is 'float'
142     results = cv2.dnn.NMSBoxes(bounding_boxes, confidences,
143                                probability_minimum, threshold)
144
145     ### Drawing bounding boxes and labels
146
147     # Checking if there is at least one detected object
148     # after non-maximum suppression
149     if len(results) > 0:
150         # Going through indexes of results
151         for i in results.flatten():
```

```
152             # Getting current bounding box coordinates,
153             # its width and height
154             x_min, y_min = bounding_boxes[i][0], bounding_boxes[i
    ][1]
155             box_width, box_height = bounding_boxes[i][2],
    bounding_boxes[i][3]
156 
157             # Preparing colour for current bounding box
158             # and converting from numpy array to list
159             colour_box_current = colours[class_numbers[i]].tolist()
160 
161             # Drawing bounding box on the original current frame
162             cv2.rectangle(frame, (x_min, y_min),
163                           (x_min + box_width, y_min + box_height),
164                           colour_box_current, 2)
165 
166             # Preparing text with label and confidence for current
    bounding box
167             text_box_current = '{}: {:.4f}'.format(labels[int(
    class_numbers[i])],
168                                                    confidences[i])
169 
170             # Putting text with label and confidence on the
    original image
171             cv2.putText(frame, text_box_current, (x_min, y_min - 5)
    ,
172                         cv2.FONT_HERSHEY_SIMPLEX, 0.5,
    colour_box_current, 2)
173 
174     ### Showing results obtained from camera in Real Time
175     cv2.namedWindow('YOLO v3 Real Time Detections', cv2.
    WINDOW_NORMAL)
176     cv2.imshow('YOLO v3 Real Time Detections', frame)
177     # Breaking the loop if 'q' is pressed
178     if cv2.waitKey(1) & 0xFF == ord('q'):
179         break
180 
181 # Destroying all opened OpenCV windows
182 cv2.destroyAllWindows()
```

CÓDIGO A.4: Código para la detección de objetos en tiempo real con arquitecturas YOLO y una cámara IP.

Apéndice B

CÓDIGOS EN C++ USADOS DURANTE EL DESARROLLO

```
/* El proposito de este codigo es brindar movimiento al robot propulsado para
    navegar sobre superficies acuaticas.
   Tambien proporciona la suficiente iluminacion mediante luces LED con base en
    lecturas de sensores de iluminacion utilizados.
*/
 /* Definicion de pines */
int LEDS = 10; // pin para encender las luces LED
int ENA = 9; // pin para habilitar el motorreductor
int IN1 = 8; // pin para indicar movimiento hacia adelante
int IN2 = 7; // pin para indicar movimiento hacia atras
int SENSOR_IZQ = A0; // pin para lectura del sensor izquierdo
int SENSOR_DER = A1; // pin para lectura del sensor derecho

/* Definicion de variables */
int intensidad_leds = 127; // intensidad luces LED
float val_sensor_izq = 0.0; // lectura del sensor izquierdo
float val_sensor_der = 0.0; // lectura del sensor derecho
float promedio_sensores = 0.0; // promedio de iluminacion leida
float factor = 1.0; // factor de iluminacion
```

```
18
19 /* Definicion de constantes */
20 float SENSOR_MAX_VAL = 950.0; // valor maximo de los sensores
21 float SENSOR_MIN_VAL = 10.0; // valor minimo de los sensores
22 float SENSOR_MID_VAL = (SENSOR_MAX_VAL-SENSOR_MIN_VAL)/2; // valor medio de
       iluminacion
23
24 void setup(){
25   // Motor
26   pinMode(ENA, OUTPUT);
27   digitalWrite(ENA, HIGH); // habilitacion del motor
28   pinMode(IN1, OUTPUT); // pin IN1 para movimiento hacia adelante
29   pinMode(IN2, OUTPUT); // pin IN2 para movimiento hacia atras
30   digitalWrite(IN1, HIGH); // motor avanzando hacia adelante
31   digitalWrite(IN2, LOW);  // sin senial hacia atras
32 }
33
34 void loop(){
35   /* lectura de los valores de los sensores */
36   val_sensor_izq = analogRead(SENSOR_IZQ);
37   val_sensor_der = analogRead(SENSOR_DER);
38   promedio_sensores = (val_sensor_izq+val_sensor_der)/2;
39   factor = SENSOR_MID_VAL/promedio_sensores; // calculo del factor de intensidad
40   if (promedio_sensores > SENSOR_MID_VAL){
41     intensidad_leds = max(intensidad_leds*factor, 1);
42   }
43   else{
44     if (promedio_sensores < SENSOR_MID_VAL){
45       intensidad_leds = min(intensidad_leds*factor, 255);
46     }
47   }
48   analogWrite(LEDS, intensidad_leds); // se proporciona iluminacion necesaria
49 }
```

CÓDIGO B.1: Código usado en el Arduino durante la implementación

Apéndice C

PLANOS DE LA ESTRUCTURA DEL ROBOT

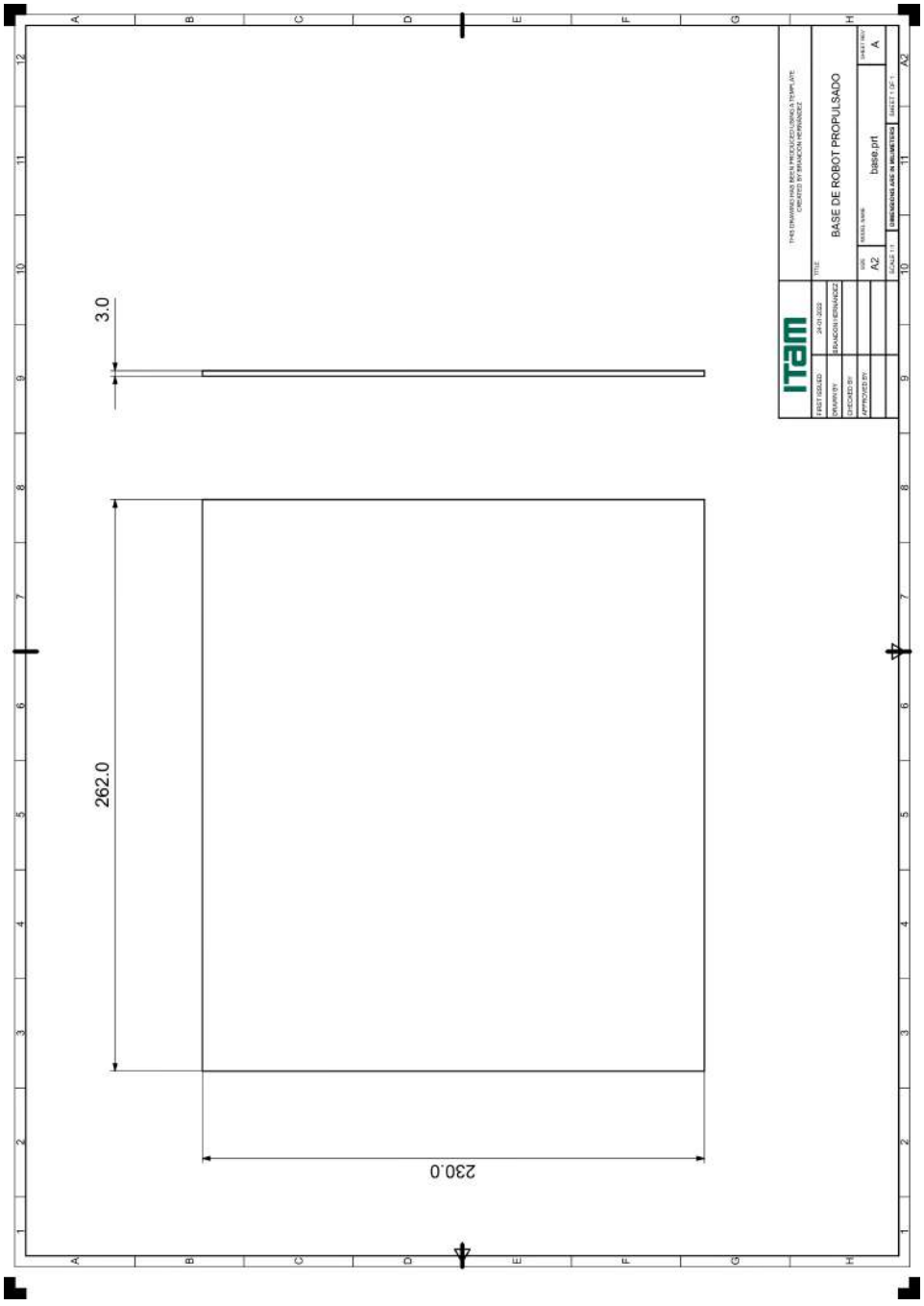


Figura C.1: Plano de la pieza *base* de Robot propulsado.

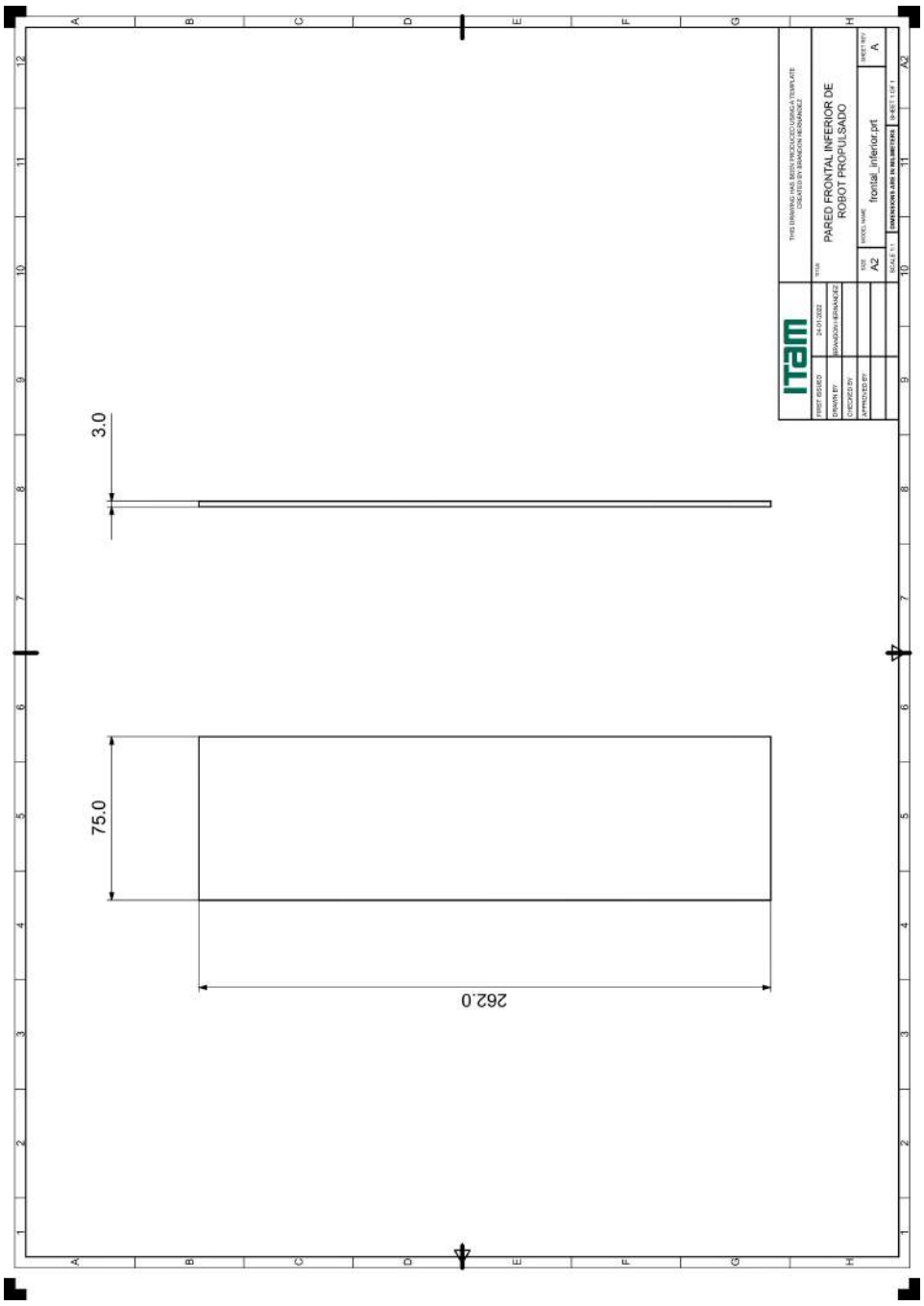


Figura C.2: Plano de la pieza *pared frontal inferior* de Robot propulsado.

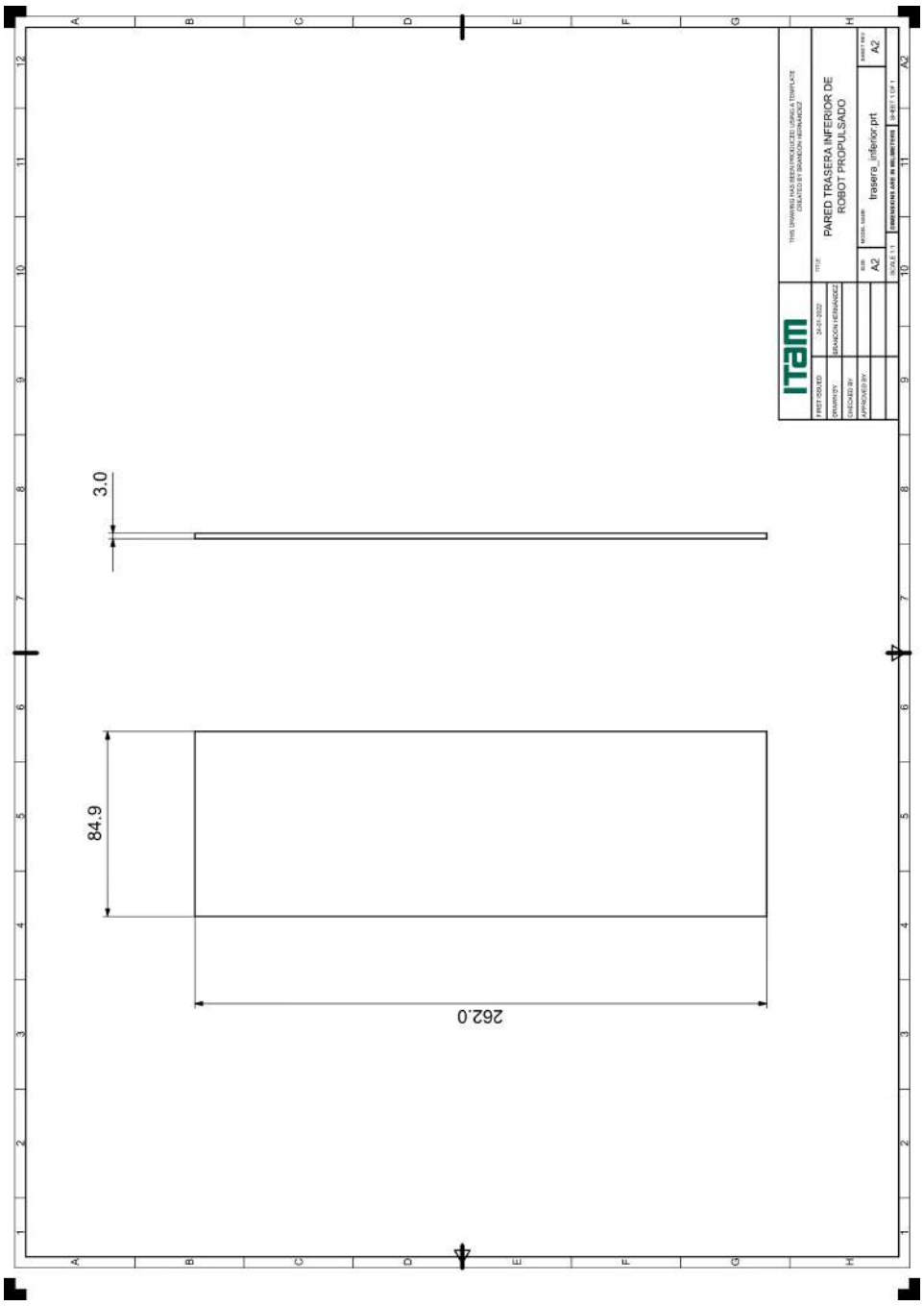


Figura C.3: Plano de la pieza *pared trasera inferior* de Robot propulsado.

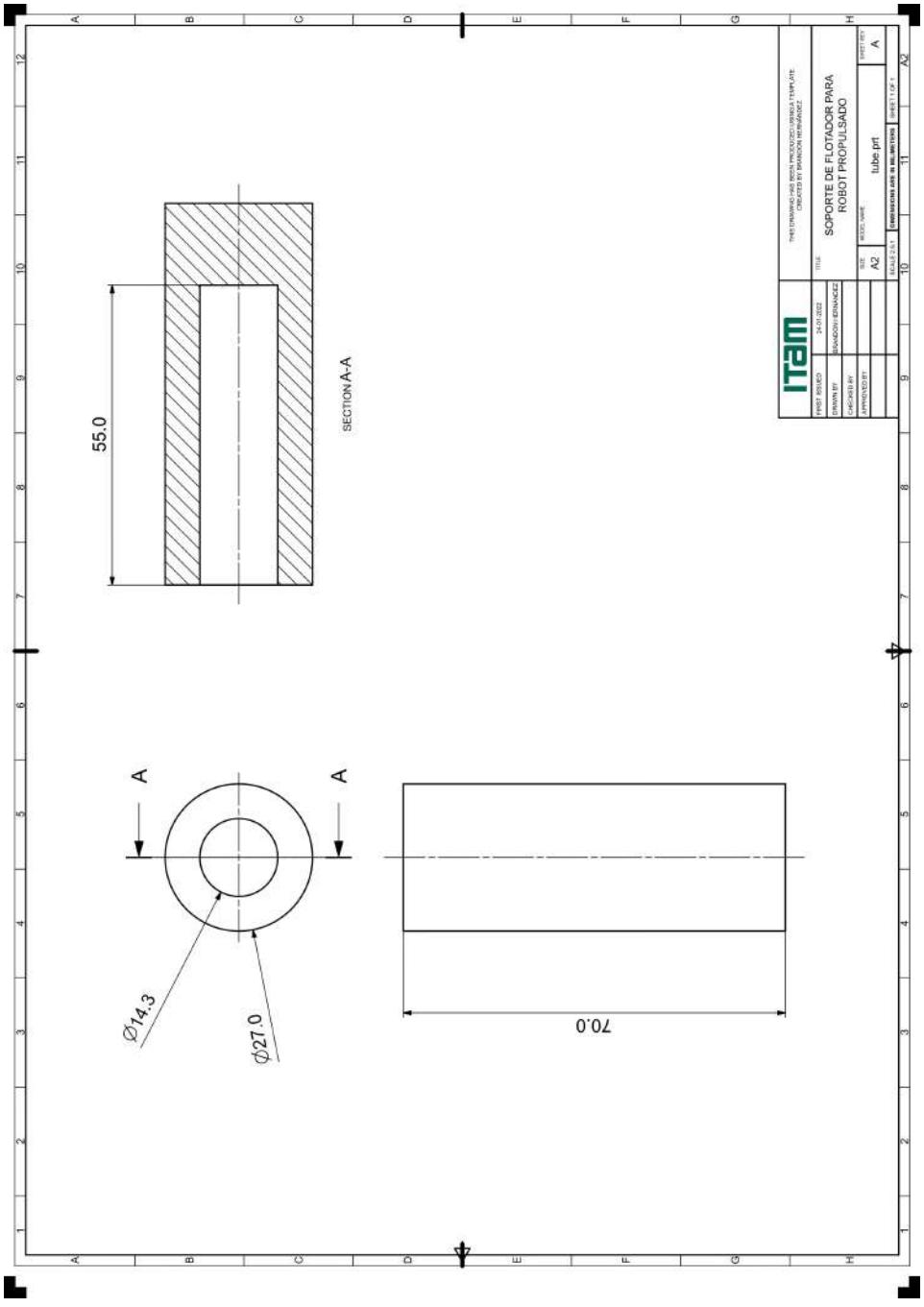


Figura C.4: Plano de la pieza *soporte* de Robot propulsado.

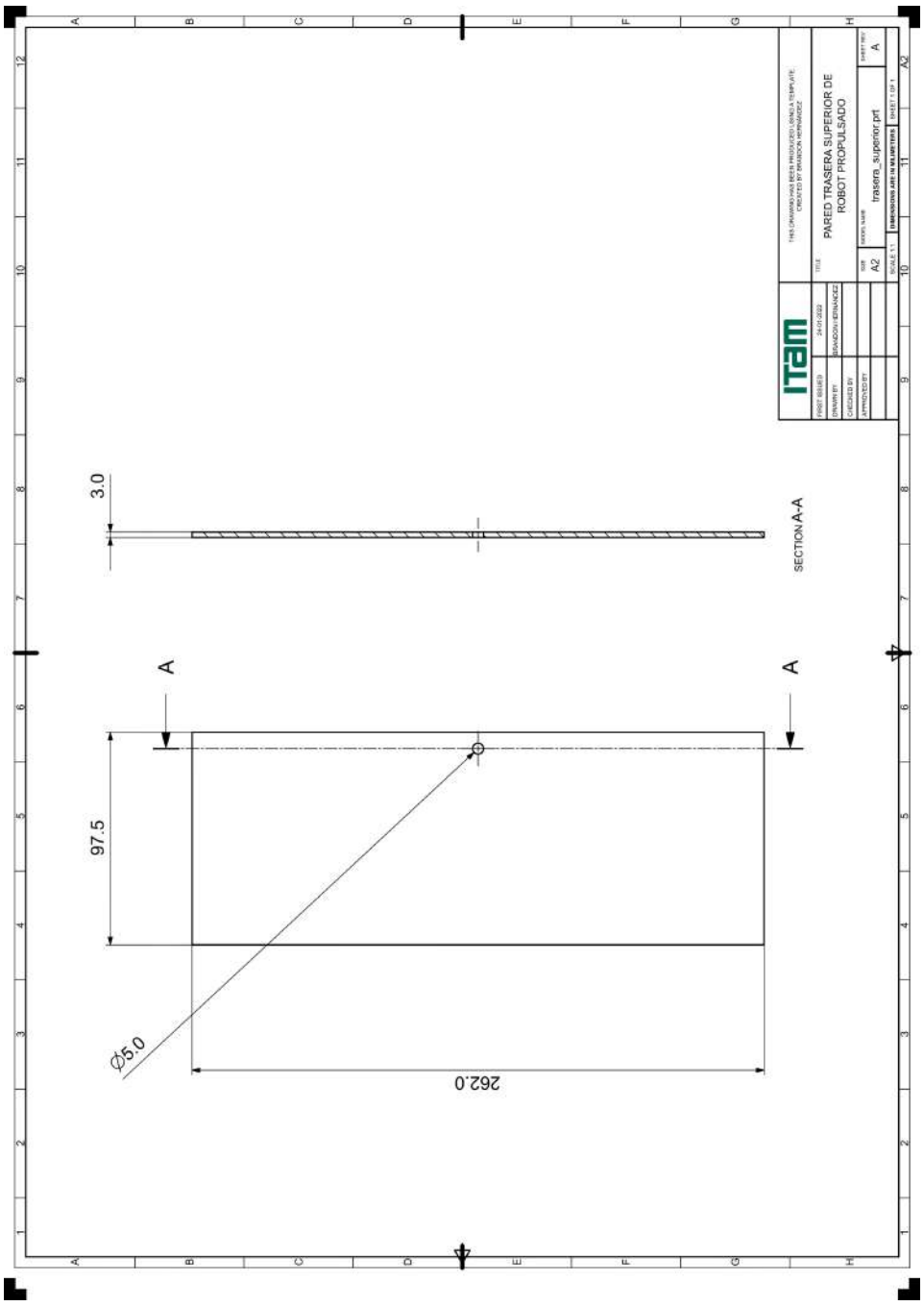


Figura C.5: Plano de la pieza *pared trasera superior* de Robot propulsado.

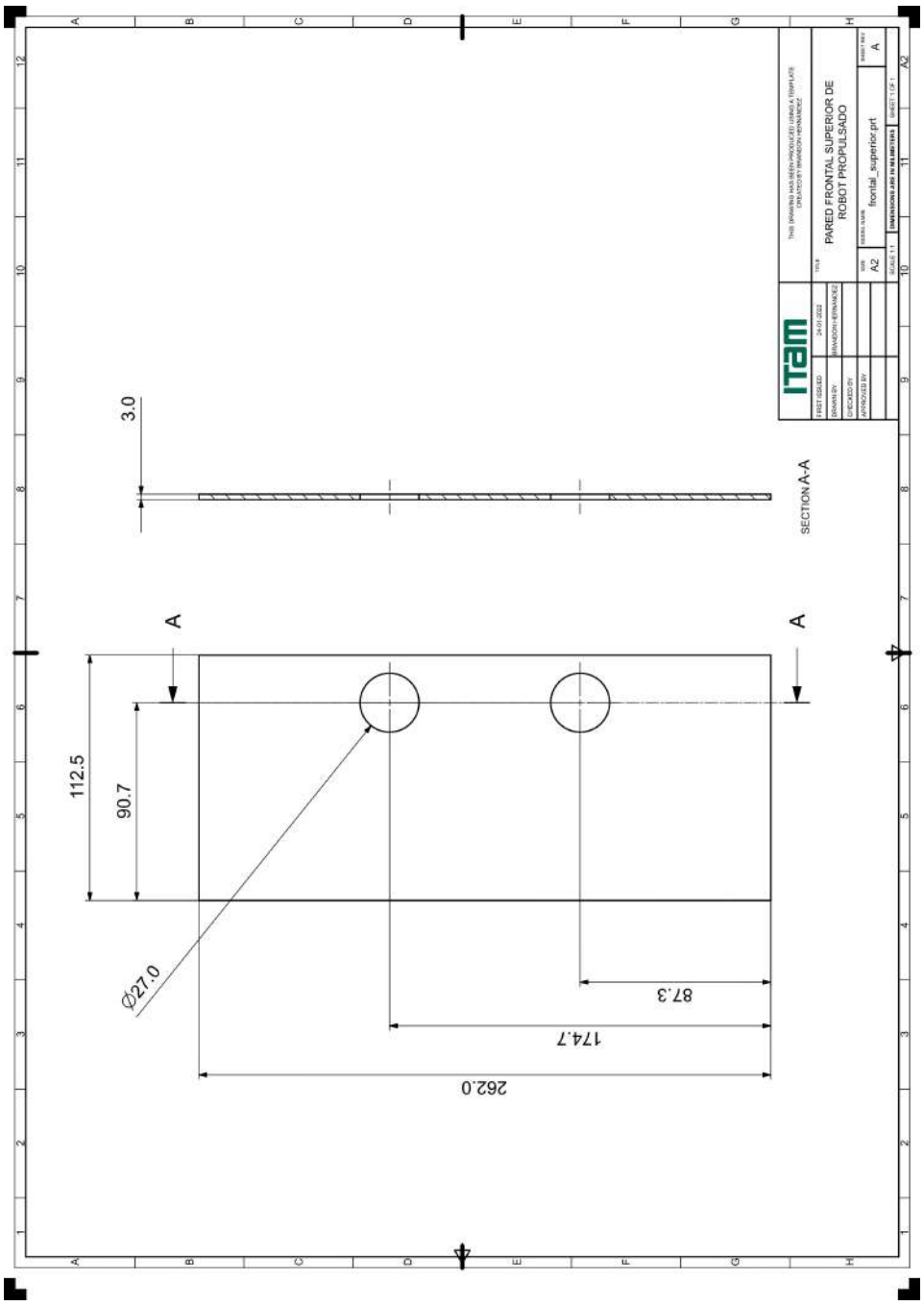


Figura C.6: Plano de la pieza *pared frontal superior* de Robot propulsado.

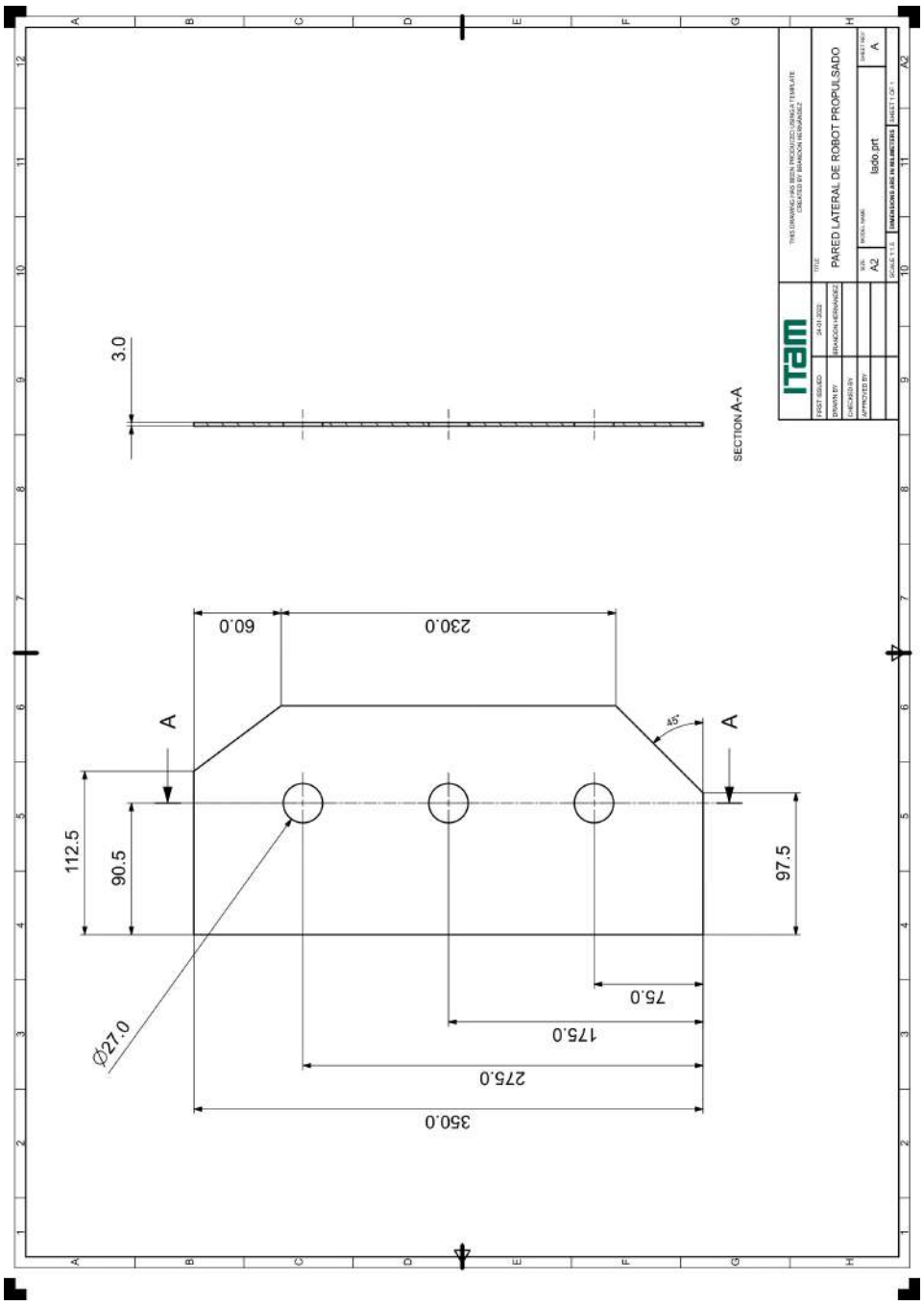


Figura C.7: Plano de la pieza *pared lateral* de Robot propulsado.

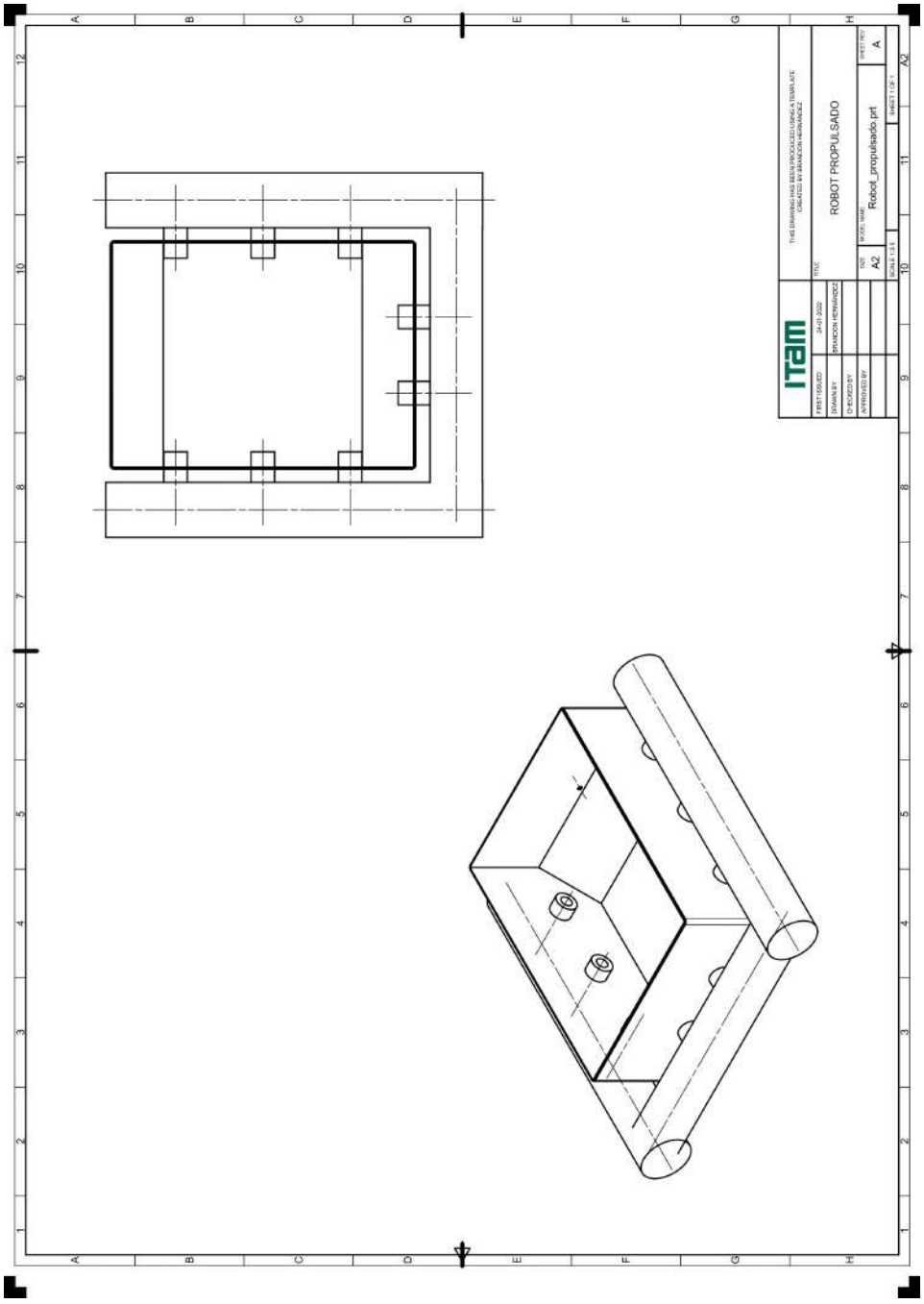


Figura C.8: Plano de la integración final de componentes de Robot propulsado.

Apéndice D

CÓDIGOS-G PARA LA MANUFACTURA DE LAS PIEZAS

```
%
O01000
M98 P1001
M98 P1002
M98 P1003
M98 P1004
M98 P1005
M05
M30
%
```

CÓDIGO D.1: Código-G para la ejecución de las operaciones.

```
%
O01001
N0010 G40 G18 G94 G80 G21
N0020 G50 S1500
N0030 T0101
(Trabajo         :  FACING)
(Fecha           :  Wed Jul 14 01:37:37 2021)
(No. de herram.  :  1)
(Nom. de herram. :  OD_80_L)
(Operacion elaborada por Brandon Hernandez)
( Inicio del programa )
N0040 G54 (Equipo leido: N/A)
N0050 G97 S1500 M03
N0060 G00 X55.186 Z11.44
N0070 X46.25 Z-.667
N0080 G50 S1500
N0090 G96 S150 M03 (Edit tool > More > X Mount = surface speed)
N0100 G01 X43.85 F.05
N0110 X-2.4
N0120 X-4.8
N0130 G00 Z2.333
N0140 X46.5
N0150 Z-1.333
N0160 G01 X44.1
N0170 X-2.4
N0180 X-4.8
N0190 G00 Z1.667
N0200 X46.5
N0210 Z-2.
N0220 G01 X44.1
N0230 X-2.4
N0240 X-4.8
N0250 G00 X55.186 Z11.44
N0260 G28 U0.
N0270 G28 W0.
N0280 M99
%
```

CÓDIGO D.2: Código-G para la operación de Careado.

```
%
O01002
N0010 G40 G18 G94 G80 G21
N0020 G50 S1500
N0030 T0606
(Trabajo         :  DRILLING)
(Fecha           :  Wed Jul 14 01:38:03 2021)
(No. de herram.  :  6)
(Nom. de herram. :  STD_DRILL)
(Operacion elaborada por Brandon Hernandez)
( Inicio del programa )
N0040 G54 (Equipo leido: N/A)
N0050 G00 X0.0 Z1.
N0060 G81 Z-61.293 R1. F334.
N0070 G80
N0080 G28 U0.
N0090 G28 W0.
N0100 M99
%
```

CÓDIGO D.3: Código-G para la operación de Perforado.

```
%
O01003
N0010 G40 G18 G94 G80 G21
N0020 G50 S1500
N0030 T0101
(Trabajo         :  ROUGH_TURN_OD)
(Fecha           :  Wed Jul 14 01:38:27 2021)
(No. de herram.  :  1)
(Nom. de herram. :  OD_80_L)
(Operacion elaborada por Brandon Hernandez)
( Inicio del programa )
N0040 G54 (Equipo leido: N/A)
N0050 G97 S1500 M03
N0060 G00 X55.186 Z11.44
N0070 X34.867 Z2.2
N0080 G50 S1500
N0090 G96 S150 M03 (Edit tool > More > X Mount = surface speed)
N0100 G01 Z1. F.05
N0110 Z-72.5
N0120 X38.1
N0130 X39.797 Z-71.651
N0140 G00 Z2.2
N0150 X31.633
N0160 G01 Z1.
N0170 Z-72.5
N0180 X34.867
N0190 X36.564 Z-71.651
N0200 G00 Z-1.151
N0210 X30.097
N0220 G01 X28.4 Z-2.
N0230 Z-72.5
N0240 X31.633
N0250 X33.33 Z-71.651
N0260 G00 X55.186 Z11.44
N0270 G28 U0.
N0280 G28 W0.
N0290 M99
%
```

CÓDIGO D.4: Código-G para la operación de Desbaste.

```
%
O01004
N0010 G40 G18 G94 G80 G21
N0020 G50 S1500
N0030 T0808
(Trabajo         :  FINISH_TURN_OD)
(Fecha           :  Wed Jul 14 01:38:39 2021)
(No. de herram.  :  8)
(Nom. de herram. :  OD_35_L)
(Operacion elaborada por Brandon Hernandez)
( Inicio del programa )
N0040 G54 (Equipo leido: N/A)
N0050 G97 S1500 M03
N0060 G00 X55.186 Z11.44
N0070 X25.3 Z-1.6
N0080 G50 S1500
N0090 G96 S100 M03 (Edit tool > More > X Mount = surface speed)
N0100 G02 X26.1 Z-2. I.4 K0.0 F.05
N0110 G03 X27. Z-2.45 I0.0 K-.45
N0120 G01 Z-71.95
N0130 G03 X26.679 Z-72.295 I-.45 K0.0
N0140 G01 X24.997 Z-73.
N0150 X38.1
N0160 G02 X38.9 Z-72.6 I0.0 K.4
N0170 G00 X55.186 Z11.44
N0180 G28 U0.
N0190 G28 W0.
N0200 M99
%
```

CÓDIGO D.5: Código-G para la operación de Acabado.

```
%
O01005
N0010 G40 G18 G94 G80 G21
N0020 G50 S1500
N0030 T0505
(Trabajo         :  PARTOFF)
(Fecha           :  Wed Jul 14 01:38:49 2021)
(No. de herram.  :  5)
(Nom. de herram. :  OD_GROOVE_L)
(Operacion elaborada por Brandon Hernandez)
( Inicio del programa )
N0040 G54 (Equipo leido: N/A)
N0050 G00 X55.186 Z11.44
N0060 X44.5 Z-75.
N0070 G97 S150 M03
N0080 G01 X44.1 F.7
N0090 X0.0
N0100 G04 P1
N0110 X.4
N0120 G00 X44.5
N0130 X55.186 Z11.44
N0140 G28 U0.
N0150 G28 W0.
N0160 M99
%
```

CÓDIGO D.6: Código-G para la operación de Corte.

REFERENCIAS

[1] E. Achar, "Basura: Causa principal de inundaciones en CDMX." *El Economista*, 2019. [Online]. Available: https://www.eleconomista.com.mx/opinion/Basura-Causa-principal-de-inundaciones-en-CDMX--20190709-0032.html

[2] E. Alpaydin, *Introduction to machine learning*, 2nd ed. MIT press, 2020.

[3] APEMAN, "Apeman a79 action camera." [Online]. Available: https://apemans.com/products/apeman-a79-action-camera

[4] Asus, "Tinker board s." [Online]. Available: https://tinker-board.asus.com/product/tinker-board-s.html

[5] M. Bennett and P. Stephan, *History of cognitive neuroscience*. Wiley Online Library, 2008.

[6] J. Bitton and Z. Papakipos, "Augly: A data augmentations library for audio, image, text, and video." https://github.com/facebookresearch/AugLy, 2021.

[7] J. Bobulski and J. Piatkowski, "Pet waste classification method and plastic waste database-wadaba." in *International Conference on Image Processing and Communications*. Springer, 2017, pp. 57–64.

[8] A. Bochkovskiy, C.-Y. Wang, and H.-Y. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv:2004.10934*, 2020.

[9] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Inc., 2008.

[10] A. Budiyono, "Advances in unmanned underwater vehicles technologies: Modeling, control and guidance perspectives." *Indian Journal of Geo-Marine Sciences*, 2009.

[11] H. Burde, "Amphibious delivery robot," GrabCAD. [Online]. Available: https://grabcad.com/library/amphibious-delivery-robot-1/details?folder_id=8979745

[12] D. Campos, *Introducción a la hidrología urbana.* Printego, 2010.

[13] CENAPRED, "Inundaciones." 2019. [Online]. Available: http://www.cenapred.gob.mx/es/Publicaciones/archivos/3-FASCCULOINUNDACIONES.PDF

[14] Compaq, Digital Equipment Corporation, IBM, Intel, Microsoft, NER, and Northern Telecom, "Universal serial bus specification," Tech. Rep., 1996.

[15] CONAGUA, "Manual para el control de inundaciones," 2011.

[16] N. Corporation, "Jetson nano developer kit." [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[17] Cámara de Diputados del H. Congreso de la Unión, "Ley de aguas nacionales." Reforma DOF 06-01-2020. [Online]. Available: http://www.diputados.gob.mx/LeyesBiblio/pdf/16_060120.pdf

[18] G. developers, "Classification: Precision and recall," 2022. [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification

[19] R. Eustice, H. Singh, J. Leonard, M. Walter, and R. Ballard, "Visually navigating the rms titanic with slam information filters." in *Robotics: Science and Systems*, vol. 2005, 2005, pp. 57–64.

[20] A. Farhadi and J. Redmon, "Yolov3: An incremental improvement," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 1804–2767.

[21] M. Fulton, J. Hong, J. Islam, and J. Sattar, "Robotic detection of marine litter using deep visual detection models." in *2019 International Conference on Robotics and Automation (ICRA).* IEEE, 2019, pp. 5752–5758.

[22] S. Garfinkel, "Usb deserves more support," *The Boston Globe*, 1999. [Online]. Available: https://web.archive.org/web/20120406080011/http://simson.net/clips/1999/99.Globe.05-20.USB_deserves_more_support+.shtml

[23] A. Géron, *Hands-On machine learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly, 2019.

[24] Goal kicker, "Python notes for professionals." online, Junio 2018.

[25] Govicture, "Victure ac700 4k cámara de acción." [Online]. Available: https://es.govicture.com/products/victure-ac700-4k-camara-de-accion

[26] S. Guzmán, "¿Por qué se inunda la Ciudad de México?" *agua.org.mx*, 2017. [Online]. Available: https://agua.org.mx/se-inunda-la-ciudad-mexico/

[27] S. Haykin, *Neural Networks: A comprehensive foundation*, 2nd ed. Pearson Education, 1999.

[28] Hitachi, Ltd., Matsushita Electric Industrial, Philips Consumer Electronics, Silicon Image, Inc., Sony Corporation, Thomson Inc., and Toshiba Corporation, "High-definition multimedia interface: Specification version 1.3a," Tech. Rep., 2006.

[29] J. Horgan and D. Toal, "Computer vision applications in the navigation of unmanned underwater vehicles." *Underwater vehicles*, pp. 195–214, 2009.

[30] T. Huang, "Computer vision: Evolution and promise." 1996.

[31] *ISO/IEC 14882:2017*, 5th ed., International Organization for Standardization, Diciembre 2017.

[32] International Telecommunications Union, International Organization for Standardization, and Joint Photographic Experts Group, "Information technology – digital compression and coding of continuous-tone still images – requirements and guidelines." Tech. Rep., 1986.

[33] Kaggle, "Garbage classification," Kaggle. [Online]. Available: https://www.kaggle.com/mostafaabla/garbage-classification

[34] P. Khlebovich, "Ip webcam," Diciembre 2021. [Online]. Available: https://play.google.com/store/apps/details?id=com.pas.webcam&hl=es_MX&gl=US

[35] H. Kondo and T. Ura, "Navigation of an auv for investigation of underwater structures." *Control engineering practice*, vol. 12, no. 12, pp. 1551–1559, 2004.

[36] V. Kotu and B. Deshpande, *Data science: concepts and practice*. Morgan Kaufmann, 2018.

[37] Y. LeCun and J. Bengio, "Convolutional networks for images, speech, and time series." *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[38] Y. LeCun *et al.*, "Generalization and network design strategies." *Connectionism in perspective*, vol. 19, pp. 143–155, 1989.

[39] Q. Liu and Y. Wu, "Supervised learning." Enero 2012.

[40] A. Maindalkar and S. Ansari, "Design of robotic fish for aquatic environment monitoring." *International Journal of Computer Applications*, vol. 117, no. 17, 2015.

[41] S. Matsumoto and Y. Ito, "Real-time vision-based tracking of submarine-cables for AUV/ROV." in *'Challenges of Our Changing Global Environment'. Conference Proceedings. OCEANS'95 MTS/IEEE*, vol. 3. IEEE, 1995, pp. 1997–2002.

[42] R. Montes, "Coladeras del DF, jugoso negocio para ladrones," *Unotv.com*, 2015. [Online]. Available: https://www.unotv.com/noticias/portal/investigaciones-especiales/detalle/delincuencia-roba-a-gdf-45-mdp-en-coladeras-406967/

[43] Z. Papakipos and J. Bitton, "Augly: A new data augmentation library to help build more robust ai models," Meta AI. [Online]. Available: https://ai.facebook.com/blog/augly-a-new-data-augmentation-library-to-help-build-more-robust-ai-models

[44] J. Pastor, "Raspberry pi 4 model b, análisis," Xataka. [Online]. Available: https://www.xataka.com/ordenadores/raspberry-pi-4-analisis-caracteristicas-precio-especificaciones

[45] J. Penalva, "Xiaomi 9t, análisis," Xataka. [Online]. Available: https://www.xataka.com/analisis/xiaomi-mi-9t-analisis-caracteristicas-precio-especificaciones

[46] B. Pollard and P. Tallapragada, "An aquatic robot propelled by an internal rotor," *IEEE/ASME Transactions on Mechatronics*, vol. 22, no. 2, pp. 931–939, 2016.

[47] R. Pressman, *Ingeniería del software: un enfoque práctico.* McGraw-Hill, 2010.

[48] P. F. Proença and P. Simões, "Taco: Trash annotations in context for litter detection," *arXiv preprint arXiv:2003.06975*, 2020.

[49] Python Software Foundation, "History and license," 2001-2020. [Online]. Available: https://docs.python.org/3/license.html

[50] A. Rácz, D. Bajusz, and K. Héberger, "Multi-level comparison of machine learning classifiers and their performance metrics," *Molecules*, vol. 24, no. 15, p. 2811, 2019.

[51] A. Ramírez, R. Castillo, B. Hernández, and A. Gómez de Silva, "A case-based approach to creating movie poster compositions," in *Proceedings of the Twelfth International Conference on Computational Creativity.* Association for Computational Creativity, 2021, pp. 266–275.

[52] Raspberry Pi Foundation, "About us," Raspberry Pi Foundation. [Online]. Available: https://www.raspberrypi.org/about

[53] J. Redmon, "Darknet: Open source neural networks in c," http://pjreddie.com/darknet/, 2013–2016.

[54] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection." in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[55] W. Roa, "Basura tapona ríos y presas en la alcaldía de Álvaro obregón," *Excelsior*, Marzo 2019. [Online]. Available: https://www.excelsior.com.mx/comunidad/basura-tapona-rios-y-presas-en-la-alcaldia-de-alvaro-obregon/1302269

[56] S. Romero, O. Romero, and D. Muñoz, *Introducción a la ingeniería*. Cernage Learning, 2015.

[57] P. Santamaría, "Gopro hero 7 black, análisis," Xataka. [Online]. Available: https://www.xataka.com/analisis/gopro-hero-7-black-analisis-caracteristicas-especificaciones-precio

[58] Y. Schechner and N. Karpel, "Clear underwater vision." in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 1. IEEE, 2004, pp. I–I.

[59] Secretaría de Protección Civil, "La basura es la causa del 50 % de las inundaciones," 2014. [Online]. Available: http://data.proteccioncivil.cdmx.gob.mx/tripticos/Triptico-Basura-2014.pdf

[60] N. Seliya, T. M. Khoshgoftaar, and J. Van Hulse, "A study on the relationships of classifier performance metrics," *21st IEEE International Conference on Tools with Artificial Intelligence*, p. 59–66, 2009.

[61] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019.

[62] V. Sichkar, "Yolo v3 objects detection with custom data," 2020. [Online]. Available: https://github.com/sichkar-valentyn/YOLO-v3-Objects-Detection-with-Custom-Data

[63] Siemens, "Nx," 2022. [Online]. Available: https://www.plm.automation.siemens.com/global/es/products/nx/

[64] Sistema Nacional de Protección Civil, "Inundaciones," Agosto 2009. [Online]. Available: http://www.proteccioncivil.gob.mx/work/models/ProteccionCivil/Resource/377/1/images/folleto_i.pdf

[65] I. SONY Latin America, "Cámara rx0 ii: pequeña, resistente y con gran calidad." [Online]. Available: https://www.sony.com.mx/electronics/camaras-compactas-cyber-shot/dsc-rx0m2

[66] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer, 2010.

[67] J. Tao, H. Wang, X. Zhang, X. Li, and H. Yang, "An object detection system based on yolo in traffic scene," in *2017 6th International Conference on Computer Science and Network Technology (ICCSNT)*, 2017, pp. 315–319.

[68] Tzutalin, "Labelimg," MIT. [Online]. Available: https://github.com/tzutalin/labelImg

[69] Ubuntu project, "About the Ubuntu project." [Online]. Available: https://ubuntu.com/about

[70] T. Van-Damme, "Computer vision photogrammetry for underwater archaeological site recording in a low-visibility environment." *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, 2015.

[71] L. Whitcomb, "Underwater robotics: Out of the research laboratory and into the field." in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 1. IEEE, 2000, pp. 709–716.

[72] Y. Xiao, "IEEE 802.11n: enhancements for higher throughput in wireless LANs." *IEEE Wireless Communications*, vol. 12, no. 6, pp. 82–91, 2005.

[73] X. Yang, T. Wang, J. Liang, G. Yao, and M. Liu, "Survey on the novel hybrid aquatic–aerial amphibious aircraft: Aquatic unmanned aerial vehicle (aquauav)," *Progress in Aerospace Sciences*, vol. 74, pp. 131–151, 2015.

[74] D. Yoerger, A. Bradley, B. Walden, M.-H. Cormier, and W. Ryan, "Fine-scale seafloor survey in rugged deep-ocean terrain with an autonomous robot," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2. IEEE, 2000, pp. 1787–1792.