
Implementación de infraestructura a escala para la evaluación de algoritmos por visión de computador para vehículos autónomos

Gabriel Alexander Fong Penagos



UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería



**Implementación de infraestructura a escala para la evaluación
de algoritmos por visión de computador para vehículos
autónomos**

Trabajo de graduación presentado por Gabriel Alexander Fong Penagos
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2024

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería



**Implementación de infraestructura a escala para la evaluación
de algoritmos por visión de computador para vehículos
autónomos**

Trabajo de graduación presentado por Gabriel Alexander Fong Penagos
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

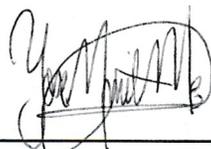
Guatemala,

2024

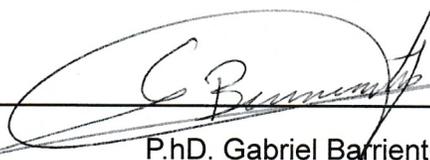
Vo.Bo.:

(f) 
M. Sc. Miguel Zea

Tribunal Examinador:

(f) 
Ing. Yosemite Meléndez

(f) 
M. Sc. Marvin Najarro

(f) 
P.h.D. Gabriel Barrientos

Fecha de aprobación: Guatemala, 13 de enero de 2024

Quiero empezar expresando mi agradecimiento a Dios en primer lugar, y a mis padres y mi hermana por brindarme el apoyo, orientación, motivación y amor durante estos años de carrera y por permitirme realizar este trabajo de graduación.

Le agradezco a mi asesor a MSc. Miguel Zea por brindarme su apoyo y conocimientos durante el proceso de elaboración del trabajo de graduación, y a cada uno de los profesores de la Universidad del Valle de Guatemala por la formación académica que me brindaron estos años.

Agradezco a mis amigos por el apoyo y convivencia durante cada uno de los momentos difíciles y desafiantes durante la pandemia.

Prefacio	III
Lista de figuras	VIII
Lista de cuadros	IX
Resumen	X
Abstract	XI
1. Introducción	1
2. Antecedentes	2
2.1. Vehículos autónomos en el mundo real	2
2.2. Vehículos autónomos a escala y visión por computador	3
2.3. Algoritmos de visión por computador en vehículos autónomos	3
3. Justificación	5
4. Objetivos	6
4.1. Objetivo general	6
4.2. Objetivos específicos	6
5. Alcance	7
6. Marco teórico	8
6.1. Visión por computadora	8
6.2. Redes neuronales convolucionales	8
6.2.1. LeNet	8
6.2.2. AlexNet	9
6.2.3. VGG	9
6.2.4. GoogleNet	10
6.3. Dataset de imágenes	10
6.3.1. CIFAR-10 y CIFAR-100	10

6.3.2. ImageNet	11
6.4. Clasificador de objetos por visión de computadora	11
6.4.1. Haar cascade	11
6.4.2. TensorFlow Lite	11
6.4.3. Edge Impulse	12
6.5. Espacios de color	12
6.5.1. Modelo RGB	12
6.5.2. Modelo CIELAB	12
6.5.3. Modelo HSI	13
6.6. Detección de líneas	13
6.6.1. Algoritmo de RANSAC en la detección de línea	14
6.6.2. Transformada de Hough para detección de línea	15
6.6.3. Regresión Theil–Sen	16
6.6.4. Algoritmo de clasificación de líneas	16
6.7. Componentes de hardware del proyecto	17
6.8. OpenMV Cam H7	17
6.8.1. Polulu 3pi+	18
6.8.2. Adafruit Neopixel	18
7. Diseño y manufactura de infraestructura vial a escala	20
7.1. Diseño de infraestructura de carriles	20
7.1.1. Diseño de carriles rectos	20
7.1.2. Diseño de carretera con esquina	21
7.1.3. Carretera con carril con intersección	23
7.1.4. Carretera con intersección de cuatro carriles	23
7.1.5. Acople de carretera	25
7.1.6. Manufactura de carretera	25
7.2. Diseño de señales de tránsito	30
7.2.1. Señal de alto	30
7.2.2. Señal de peatón	31
7.2.3. Señal de límite de velocidad	31
7.2.4. Manufactura de señales de tránsito	32
7.3. Diseño de semáforo	33
7.3.1. Programación de los semáforos	36
8. Detección de infraestructura vial mediante visión de computadora	39
8.1. Detección de carriles mediante visión de computadora	39
8.2. Haar cascade	41
8.2.1. Detección de señal de alto	41
8.2.2. Detección de vehículos	42
8.3. TensorFlow Lite	43
8.3.1. Edge Impulse	43
8.3.2. Requisitos para utilizar TensorFlow Lite en OpenMV	47
8.3.3. Detección de señales de tránsito y estados de semáforo	50
8.3.4. Protocolo de comunicación	53

9. Validación	55
9.1. Validación de modelos de detección de objetos .tflite	55
9.2. Validación de la detección de carriles	58
10. Conclusiones	63
11. Recomendaciones	64
12. Bibliografía	65
13. Anexos	68
13.1. Diseños de infraestructura a escala	68
13.2. Detección de carriles en escenas de carreteras	68
13.3. Dataset de señales de tránsito	68
13.4. Repositorio de códigos de los algoritmos de visión por computados	68
14. Glosario	69

Lista de figuras

1.	Ejemplo de detección de vehículos y señales de tránsito [6].	4
2.	Ejemplo de detección de carril y superficies [6].	4
3.	Algoritmo de LeNet [9].	9
4.	Algoritmo de Alexnet [9].	9
5.	Base de datos de CIFAR10 [12].	10
6.	Ejemplo de detección de bordes en carriles utilizando el algoritmo de Canny [20].	14
7.	Ejemplo de detección de líneas de carriles utilizando el algoritmo de RAN-SAC [7].	15
8.	Ejemplo de detección de líneas de carriles utilizando la transformada de Hough [20].	16
9.	OpenMV Cam H7 de OpenMV [22].	17
10.	Diseño de Polulu 3pi+ con dimensiones [23].	18
11.	Matriz de 64 Neopixel [24].	19
12.	Diseño a escala de carretera de dos carriles, vista superior.	21
13.	Diseño a escala de esquina de dos carriles primera versión, vista superior.	22
14.	Diseño a escala de esquina de dos carriles segunda versión, vista superior.	22
15.	Diseño a escala de intersección de dos carriles, vista superior.	23
16.	Diseño a escala de cruce de cuatro carriles, vista superior.	24
17.	Diseño a escala del ensamble carreteras, esquinas, intersecciones y cruce de cuatro carriles, vista superior.	24
18.	Diseño de acople de carretera a escala, vista isométrica.	25
19.	Carretera a escala manufacturada.	26
20.	Acople de carretera manufacturada.	27
21.	Acople y carretera a escala manufacturada.	28
22.	Esquina a escala manufacturada.	28
23.	Circuito de carretera armado en la plataforma del Robotat.	29
24.	Dimensiones de altura de señales de tránsito en área urbana [25].	30
25.	Diseño de dimensiones de señal de alto [25].	31
26.	Diseños 3D finales de las señales de tránsito.	32
27.	Diseños manufacturados finales de las señales de tránsito.	33
28.	Diseño de dimensiones de altura de semáforo [25].	34

29. Diseño 3D de semáforo.	34
30. Conector JST-SM de 3 pines [26]	35
31. Diseños manufacturados final del semáforo a escala.	36
32. Resultado de aplicación de algoritmo de detección de línea en carretera a escala en el IDE de OpenMV.	40
33. Resultado de aplicación de algoritmo de detección de línea en carretera a escala.	41
34. Resultado de aplicación de algoritmo de detección de señal de alto con <i>Haar</i> <i>cascade</i>	42
35. Resultado de aplicación de algoritmo de detección vehículo con <i>Haar cascade</i>	43
36. Sección <i>data acquisition</i> en la plataforma de <i>Edge impulse</i>	44
37. Etiquetado de semáforo en verde y rojo en la plataforma de <i>Edge impulse</i>	44
38. Resultados de etiquetado de <i>dataset</i> en la plataforma de <i>Edge impulse</i>	45
39. Configuraciones de bloques para el proceso de entrenamiento del modelo en la plataforma de <i>Edge impulse</i>	45
40. Configuraciones de red neuronal para el proceso de entrenamiento del modelo en la plataforma de <i>Edge impulse</i>	46
41. Selección de librería para exportar el modelo en la plataforma de <i>Edge impulse</i>	47
42. Archivos exportados del modelo de entrenamiento.	47
43. Selección de <i>firmware</i> a utilizar en la plataforma de <i>Edge impulse</i>	48
44. Selección del <i>firmware</i> a utilizar en el IDE de OpenMV.	49
45. Selección de ruta del <i>firmware</i> a utilizar en el IDE de OpenMV.	49
46. Mensaje de <i>firmware</i> actualizado en el IDE de OpenMV.	49
47. Detección de la señal de alto en el IDE de OpenMV.	51
48. Detección de señal de alto.	52
49. Detección de la señal de peatón en el IDE de OpenMV.	52
50. Detección de señal de peatón.	53
51. Detección de semáforos.	53
52. Matriz de confusión del primer modelo de entrenamiento de 128x128 píxeles.	56
53. Matriz de confusión del segundo modelo de entrenamiento de 96x96 píxeles.	56
54. Matriz de confusión del tercer modelo de entrenamiento de 96x96 píxeles.	57
55. Detección de carriles en carretera hacia Utah.	59
56. Detección de carriles en carretera hacia el parque nacional Yosemite	59
57. Detección de carriles en carretera hacia la ruta estatal 123 en el parque na- cional Pinchot.	60
58. Detección de carriles en carretera de noche en la ciudad de Seúl.	60
59. Fotogramas del algoritmo detección de carril amarillo implementado.	61
60. Diagrama de solución planteada.	62

Lista de cuadros

1. Tabla de dimensiones escaladas	20
2. Tabla de piezas manufacturadas	26
3. Tabla de dimensiones de la señal de alto [25].	31
4. Tabla de dimensiones de señales de rectangulares y prevención [25].	32
5. Tabla de piezas manufacturadas	57
6. Tabla de piezas manufacturadas	58
7. Tabla de piezas manufacturadas	58

La siguiente propuesta de trabajo tiene como objetivo principal desarrollar infraestructura a escala para realizar pruebas de visión de computador en un vehículo autónomo a escala, en donde se utilizará como base de la escala de vehículo el Polulu 3Pi+. Para ello, se utilizó una escala de reducción de 1:18.75 para el diseño y la manufactura de infraestructura vial a escala, incluyendo carreteras rectas, con curvas, señales de tránsito y semáforos. Las piezas se fabricaron de material MDF, mediante corte láser.

Se utilizó la OpenMV Cam H7 para la visión de computador, en la que se implementó un algoritmo de detección de carriles de color amarillo y alineación verticalmente. El objetivo de este algoritmo es brindar direccionamiento al vehículo.

Adicionalmente, se implementaron modelos de detección y clasificación de señales de tránsito y estados de los semáforos. Se utilizó la plataforma de Edge Impulse para entrenar y optimizar un modelo de *machine learning*, en donde se obtuvo un modelo TensorFlow Lite de detección de 5 clases de objetos (señal de alto, señal de peatón y estados del semáforo), con un tamaño de 56 KB. El modelo de entrenamiento demostró ser capaz de reconocer cada una de las clases entrenadas.

Computer vision plays an important role in the operation of autonomous vehicles. One of the roles is the detection and classification of objects in the environment, such as vehicles, traffic signs, traffic lights, people, and more. Another relevant role is environment perception, where road lanes are identified, enabling autonomous vehicle mobility.

In this graduation project, the overall objective is to develop the infrastructure at a scale suitable for conducting computer vision experiments on a scaled autonomous vehicle. The design and manufacturing of scaled road infrastructure were carried out, including two-lane roads, traffic signs, and traffic lights.

Line detection algorithms were implemented for lane detection using the OpenMV Cam H7, where the use of an RGB565 color filter and the Hough transform allowed us to detect the the yellow line.

Object detection algorithms were also implemented, where we managed to detect and classify the states of traffic lights, vehicles, traffic signs such as stop signs and pedestrian signs, where TensorFlow Lite was used on the Edge Impulse platform.

La visión por computadora desempeña un papel importante en el funcionamiento de los vehículos autónomos. Uno de sus roles principales es la detección y clasificación de objetos en el entorno, como vehículos, señales de tránsito, semáforos, personas y más. Otro rol relevante es la percepción del entorno, donde se identifican los carriles de las vías, lo que permite la movilización autónoma del vehículo.

En el presente trabajo de graduación el objetivo general es desarrollar la infraestructura a escala de un ecosistema adecuado para realizar experimentos de visión de computadora en un vehículo autónomo a escala. En el capítulo 7 se realizó el diseño y la manufactura de infraestructura vial a escala, en donde se realizaron carreteras de dos carriles, señales de tránsito y semáforos.

Luego, en el capítulo 8 se implementaron algoritmos de detección de línea para los carriles utilizando la OpenMV Cam H7, donde el uso de un filtro de color RGB565 y la transformada de Hough permitieron la detección de la línea de color amarillo.

Finalmente se implementaron algoritmos de detección de objetos, en donde se logró detectar y clasificar estados de semáforos, vehículos, señales de tránsito como señal de alto y de peatón, con base en TensorFlow Lite dentro de la plataforma de Edge Impulse.

2.1. Vehículos autónomos en el mundo real

Los vehículos autónomos son capaces de operar sin la intervención humana en tareas como la navegación, toma de decisiones y control, donde se definen seis niveles de autonomía, desde el nivel 0 cuando el vehículo no presenta ningún tipo de autonomía, hasta el nivel 5, en donde el vehículo presenta autonomía total y no es necesario de la intervención humana. La norma ISO 26262 establece requisitos específicos para los vehículos autónomos en donde se requiere la validación y verificación de conducción, las cuales pueden estar limitadas por visibilidad, ambiente externo, velocidad y acceso a carreteras o en caminos de área rural. Esta norma establece que debe de existir un método de operación para cualquiera de los casos y para cualquier falla eventual que ocurra en el vehículo o en su entorno, se han utilizado métodos como inductive learning y Fail-Operational System, pero no han mostrado ser certeros en cada una de las situaciones [1].

El Autonomous Vehicles Readiness Index (AVRI) es un índice que muestra la preparación de los países para el despliegue de los vehículos autónomos, el cual evalúa 28 distintos aspectos los cuales se basan en 4 pilares: Política y legislación, tecnología e innovación, infraestructura y aceptación del consumidor. En infraestructura se toma en cuenta la calidad de las carreteras, puestos de cargar para vehículos eléctricos, las señales de tránsito, tecnología para mejorar el tránsito vehicular. En el área de aceptación en el consumidor evalúan la utilidad, aceptación y pruebas en distintas áreas de los vehículos autónomos. Los tres países con mejores índices son: Singapur, Países Bajos y Noruega, siendo Singapur el primero en la lista por su alta inversión en infraestructura y teniendo una respuesta positiva en los consumidores. China, con el puesto 20, ha decidido realizar pruebas en la ciudad de Beijing con los vehículos autónomos nivel 5, colocando también nuevas leyes y colocando un solo carril en carreteras de alta velocidad para los vehículos autónomos. India tiene el puesto 29 siendo el penúltimo en el índice, mostrando bajos índices en cada uno de los 4 pilares, pero también buscan desarrollar algoritmos que se adecuen de mejor manera su entorno [2].

2.2. Vehículos autónomos a escala y visión por computador

En la Universidad de San Francisco de Quito (USFQ) realizaron un vehículo autónomo, con autonomía nivel 3. Como base del proyecto se utilizó un BMW i8 Spyder Toy Roll Play, como el núcleo de control se utilizó un Raspberry 4 y el Nvidia Jetson Nano para el uso de inteligencia artificial para la clasificación y reconocimiento de los objetos. Limitaron el reconocimiento de objetos a nueve, los cuales fueron bicicletas, vehículos, buses, motocicleta, luces de tráfico, señales de tráfico, personas y señales de alto. Tuvieron limitaciones con la resolución y actualización de las imágenes en tiempo real, llegando a un máximo de 15 FPS, recomendando encontrar una alternativa al no poder utilizar OpenCV y utilizar componentes que se puedan comunicar mediante ROS [3].

En el Instituto Tecnológico Superior de Atlixco realizaron el diseño de un vehículo autónomo para la competencia AutomodelCar para el Torneo Mexicano de Robótica, donde se utilizó el ecosistema ROS y la librería OpenCV para el entrenamiento y procesamiento de las imágenes. El controlador del sistema fue el Nvidia Jetson Nano y la cámara utilizada fue el modelo Sony IMX219. Los aspectos a considerar fueron que la superficie era una lona negra y las líneas de carril eran de color blanco con 40cm de ancho, las condiciones de la luz podían estar entre 300 a 500 lux, siendo estos variables para distintas pruebas. Realizaron tres nodos para el procesamiento de las imágenes, el primer nodo se encargaba de la detección de las imágenes, el segundo nodo se encargaba de la detección de las líneas de carril y el tercer nodo era el encargado de la detección de intersecciones [4].

En la Universidad Politécnica Salesiana de Ecuador realizaron un robot móvil para la siembra de semillas en el campo utilizando visión por computadora. Como el controlador principal utilizaron un Raspberry Pi Pico y acoplado el módulo OpenMV Cam H7 para la visión artificial. Para el entrenamiento de la red neuronal utilizaron 754 imágenes para identificar surcos de siembra y utilizaron la arquitectura Mobile Net. Se obtuvo un 98.7 de porcentaje de acierto en las pruebas de detección y siembra de semillas en los surcos [5].

2.3. Algoritmos de visión por computador en vehículos autónomos

Para la detección de señales de tránsito, vehículos, motocicletas se recomienda detectarlas como imágenes en 2D y no en 3D, esto se debe a que se tiene mayor precisión de detección en 2D. Para la detección de objetos en 3D se necesita detectar una superficie horizontal, en donde se apoyan los objetos detectados. Para la predicción de distancia entre los objetos detectados puede utilizarse filtros de Kalman (Figura 1). Para las imágenes en 2D se crean clases para entrenar el algoritmo y luego poder detectarla, utilizando un banco de imágenes positivas y negativas. Para las superficies en 3D se utiliza el algoritmo de RANSAC, en donde se seleccionan 3 puntos del espacio, se realiza una estimación de mínimos cuadrados de los puntos anteriores, y se utiliza una máscara para detectar las líneas de los carriles utilizando un detector de bordes, en donde se remueven las líneas que no se utilizan y también se definen sólo las líneas dentro de la superficie y no toma en cuenta las que se encuentran fuera de ella (Figura 2) [6].

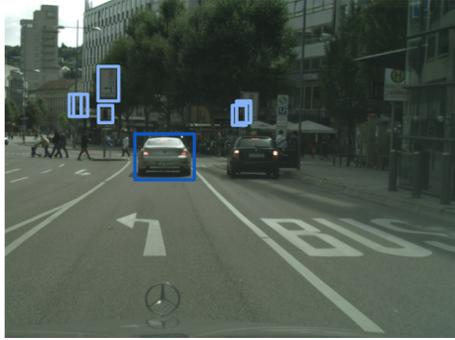


Figura 1: Ejemplo de detección de vehículos y señales de tránsito [6].



Figura 2: Ejemplo de detección de carril y superficies [6].

Los vehículos autónomos han demostrado ser efectivos en los ámbitos de detección de objetos, otros vehículos, señales de tránsito, semáforos, carriles y personas. También en los ámbitos de conducción y movilidad autónoma se comportan de manera eficiente, reduciendo la cantidad de accidentes viales. Este buen funcionamiento no depende únicamente de la tecnología del vehículo autónomo, ya que el ecosistema en donde se encuentra también influye en el comportamiento del mismo, siendo las ciudades que han obtenido mejor aceptación de los vehículos autónomos aquellas que han invertido en la infraestructura vial requerida. En Centroamérica no se han realizado validaciones del comportamiento de los vehículos autónomos, lo cual es relevante al tener distintas condiciones en los ecosistemas viales que las principales ciudades europeas y de Estados Unidos.

Tomando esto en consideración, en este proyecto se establecerá una base para la infraestructura vial a escala para la experimentación con vehículos autónomos escala, desarrollando carriles, señales de tránsito y semáforos. Además, se buscará entrenar y validar los algoritmos de visión por computadora entrenados previamente con objetos reales y a escala, utilizando el módulo OpenMV Cam H7, iniciando un nuevo campo de investigación en UVG y el país.

4.1. Objetivo general

Desarrollar la infraestructura a escala de un ecosistema adecuado para realizar experimentos de visión de computadora en un vehículo autónomo a escala.

4.2. Objetivos específicos

- Diseñar y manufacturar elementos de infraestructura vial a escala funcionales, como señales de tránsito, carriles y semáforos, que sean detectables por los algoritmos entrenados con los elementos de tamaño real y viceversa.
- Implementar y validar algoritmos de visión de computadora para vehículos autónomos utilizando el módulo OpenMV Cam H7.
- Validar la detección de los elementos a escala desarrollados y objetos reales, independientemente de su entrenamiento.

El alcance de este trabajo se centró en el diseño y fabricación de una infraestructura vial a escala para llevar a cabo experimentos de visión por computadora utilizando la OpenMV Cam H7.

Durante el estudio, se crearon modelos de entrenamiento de imágenes, los cuales fueron transformados a versiones compatibles con *TensorFlow Lite* y Cascade para su implementación en la OpenMV Cam H7. Los archivos Cascade entrenados que se utilizaron, se obtuvieron de librerías de Github. Para el procesamiento de entrenamiento y *Transfer learning* de cinco clases distintas de objetos, se empleó la plataforma de Edge Impulse. Se utilizó la versión gratuita, la cual permitió realizar dos modelos de entrenamiento por *dataset*. Esto se debió a que la versión gratuita permite utilizar cuatro horas de procesamiento en cada proyecto.

Se realizaron experimentos de detección de líneas y objetos en esta infraestructura a escala para validar los algoritmos entrenados previamente. Sin embargo, se encontró una limitación debido al espacio limitado en la RAM de la OpenMV Cam H7, que es inferior a 400 KB. Esta limitación impide cargar archivos *.cascade* o *.tflite* que excedan este tamaño, lo que a su vez impide la creación de más clases o la construcción de bibliotecas con una mayor cantidad de imágenes.

6.1. Visión por computadora

La visión por computadora realiza procesamiento de imágenes o vídeos para interpretar, analizar, extraer información y comprender lo que sucede en el mundo visual. Esto con el fin de realizar tareas de manera automática. Es aplicada en campos de procesamiento de imágenes médicas, inspección de maquinaria, construcción de modelos 3D, conducción autónoma de vehículos, captura de movimiento, reconocimiento biométrico, entre otros [7].

6.2. Redes neuronales convolucionales

Son una clase de redes neuronales multicapa *feedforward* diseñadas para el reconocimiento y clasificación de imágenes. Las redes convolucionales siguen cierta estructura, conformada por tres capas principales: Convolutional Layer, Pooling Layer y Fully-Connected Layer [8].

6.2.1. LeNet

Es una red neuronal convolucional profunda capaz de detectar caracteres utilizando conceptos de *backpropagation* y *feedforward*, en donde tiene gran cantidad de capas, numerosos mapas de unidades replicadas en cada etapa y agrupación de las salidas cuyas unidades replicadas estuvieran cercanas, en la Figura 3 se puede observar el diagrama del algoritmo utilizado por LeNet [9].

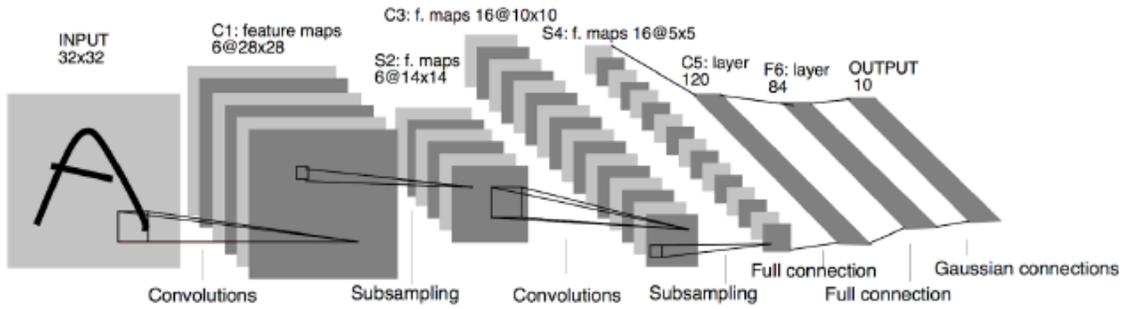


Figura 3: Algoritmo de LeNet [9].

6.2.2. AlexNet

Introducida en 2012, ha sido una de las primeras CNN con rendimiento excelente en la aplicación de visión por computador. Tiene una arquitectura similar a LeNet, siendo AlexNet más profunda, con mayor cantidad de filtros por capa y con capas convolucionales apiladas. La arquitectura mostrada en la Figura 4 contiene 8 capas, en donde las primeras cinco son convolucionales y las restantes se encuentran totalmente conectadas. La red tiene 62.3 millones de parámetros y necesita 1.1 millones de unidades de computación en una pasada hacia adelante, en la Figura 4 se puede observar el diagrama del algoritmo utilizado por AlexNet [9].

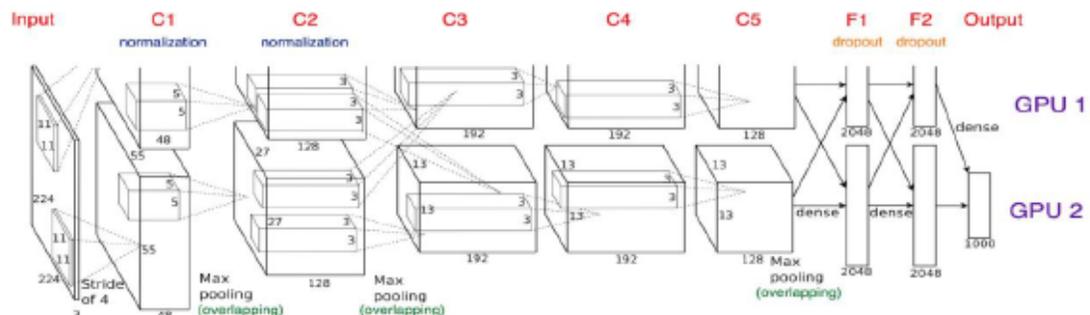


Figura 4: Algoritmo de Alexnet [9].

6.2.3. VGG

Esta red fue diseñada por el Visual Geometry Group de la universidad de Oxford. Es una red similar a AlexNet, pero con sólo capas de convolución 3x3 y numerosos filtros. Existen varias redes bajo este nombre, pero difieren en el numero de capas que poseen, como VGG-16, la cual está compuesta de 16 capas, 13 de ellas de convolución, 2 capas totalmente conectadas y una capa final de *softmax* para clasificar [9].

6.2.4. GoogleNet

Desarrollada por Google en 2014. GoogleNet está compuesta por 22 capas y tiene 12 veces menos parámetros que AlexNet, lo cual la hace más rápida y precisa. Tiene unas capas al inicio, las cuales son bloques de operaciones convolucionales de diferentes tamaños de filtro (1x1,3x3,5x5) ejecutadas en paralelo, las cuales capturan características en distintas escalas al mismo tiempo para luego combinarlas. También cuenta con capas de *pooling* y de normalización para mejorar su eficiencia[10].

6.3. Dataset de imágenes

Los dataset son colecciones de imágenes digitales, las cuales son utilizadas para evaluar y entrenar algoritmos de visión de computador. Contienen imágenes de diversos tipos, las cuales se encuentran organizadas jerárquicamente y agrupadas por clases. Mientras mayor cantidad y diversidad de imágenes contenga, mejor será el reconocimiento del modelo a utilizar[11].

6.3.1. CIFAR-10 y CIFAR-100

CIFAR-10 consiste en una base de datos de 60,000 de 32x32 píxeles imágenes a color, con 10 clases, cada clase tienen 6000 imágenes, tiene 50000 imágenes de entrenamiento y 10000 para pruebas, en la Figura 5 se logra observar un ejemplo de la base de datos de CIFAR-10. CIFAR 100 tiene 100 clases con 600 imágenes en cada clase, también tiene 20 superclases en donde 500 son de entrenamiento y 100 para pruebas[12].

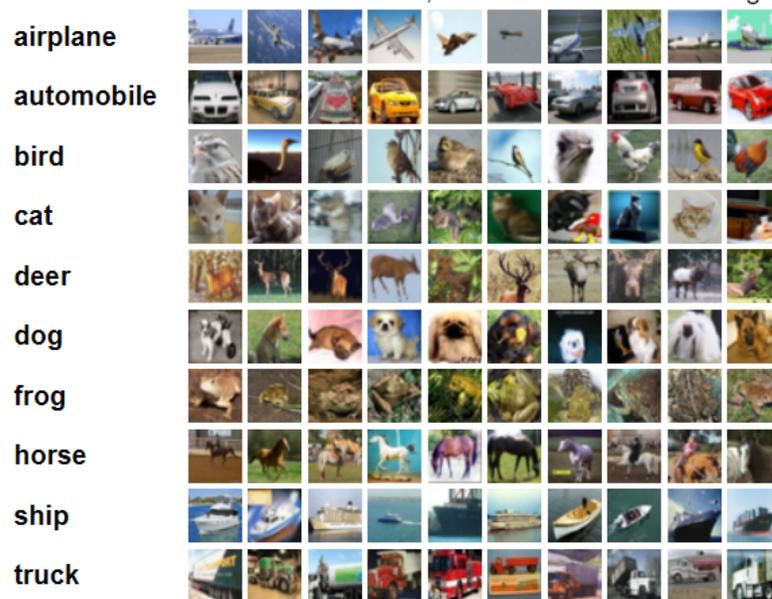


Figura 5: Base de datos de CIFAR10[12].

6.3.2. ImageNet

Es una base de datos de imágenes realizada en la universidad de Stanford con el objetivo de tener un recurso amplio y poder reconocer objetos a gran escala por medio de la visión por computador. Actualmente cuenta con más de 21000 clases o *synset* como le denominan, con 1000 imágenes en cada uno. También realizan competencias anuales para evaluar algoritmos de reconocimiento y detección de objetos por medio de visión de computador llamada *The ImageNet Large Scale Visual Recognition Challenge*(ILSVRC)" [13].

6.4. Clasificador de objetos por visión de computadora

6.4.1. Haar cascade

Los *Haar cascade* se crean entrenando el algoritmo generador con imágenes etiquetadas como positivas y negativas. Las imágenes positivas son en las que se encuentra el objeto que se desea detectar, mientras que las imágenes negativas son todo lo contrario. En las imágenes negativas, es necesario que no tengan ningún elemento en común con las imágenes positivas. Además, cuantas más imágenes se utilicen, mejores serán los resultados obtenidos. [14]

El método de *Haar cascade* es una serie de comprobaciones de contraste que se utilizan para determinar si un objeto está presente en la imagen. Estas comprobaciones de contraste se dividen en etapas, donde una etapa solo se ejecuta si las etapas anteriores ya han pasado. Las comprobaciones de contraste son cosas simples, como verificar si el centro vertical de la imagen es más claro que los bordes. Las comprobaciones de áreas grandes se realizan primero en las etapas iniciales, seguidas de comprobaciones de áreas más numerosas y pequeñas en etapas posteriores [15].

6.4.2. TensorFlow Lite

TensorFlow Lite es un conjunto de herramientas que permite el aprendizaje automático en dispositivos, facilitando que los desarrolladores ejecuten sus modelos en dispositivos móviles, embebidos y microcontroladores. *TensorFlow Lite* está optimizado para el aprendizaje automático en dispositivos, abordando 5 limitaciones clave: latencia, privacidad, no se requiere conectividad a Internet, modelo y tamaño binario reducido y consumo de energía. Además, se puede utilizar con distintos lenguajes que incluyen Java, Swift, Objective-C, C++ y Python [16].

El módulo `tf` es capaz de ejecutar modelos de TensorFlow Lite en la cámara OpenMV. El modelo `.tflite` de salida final puede cargarse directamente y ejecutarse en una cámara OpenMV. Dicho esto, el modelo y la memoria RAM temporal requerida por el modelo deben ajustarse al espacio disponible en la pila de búfer de fotogramas en la cámara OpenMV. En el caso del modelo OpenMv Cam H7 espacio disponible de memoria RAM es de 496 KB pero recomiendan que el tamaño del modelo entrenado sea menor a 400 KB [17].

6.4.3. *Edge Impulse*

Edge Impulse es una plataforma de desarrollo de *machine learning* la cual permite crear, entrenar y optimizar modelos de aprendizaje con datos del mundo real, ya sean para reconocer sonidos, identificar objetos o detectar movimiento. La plataforma ofrece bloques de preprocesamiento y aprendizaje optimizados, diversas arquitecturas de redes neuronales y modelos preentrenados, y puede generar binarios, para luego ser utilizados [18].

6.5. Espacios de color

Los espacios de color proporcionan un método para especificar, ordenar y manipular colores. Estas representaciones se corresponden con n-dimensional ordenaciones de las sensaciones de color. Los colores se representan mediante puntos en estos espacios. La gran mayoría de ellos se han desarrollado para aplicaciones específicas, aunque todos parten de un mismo concepto: la teoría tricromática de colores primarios rojo, verde y azul [19].

6.5.1. Modelo RGB

En el espacio RGB el color aparece especificado mediante cantidades positivas de rojo, verde y azul. El rango de cada coordenada o componente cromática RGB suele ser $[0,1]$ aunque en multimedia y procesamiento de imágenes está más extendida la especificación en cantidades discretas presentes en el intervalo $[0,255]$. Todos las coordenadas que se extienden en la línea que parte del punto negro al blanco corresponden a la escala de los grises [19].

Las imágenes en el modelo de color RGB están formadas por tres planos de imágenes independientes, cada uno correspondiente a un color primario. El empleo del modelo RGB para el procesamiento de imágenes es útil cuando éstas vienen expresadas en términos de los tres planos de colores. Alternativamente, la mayoría de las cámaras en color que se usan para adquirir imágenes digitales utilizan el formato RGB [19].

6.5.2. Modelo CIELAB

Los modelos de color CIELAB y CIELUV son espacios estandarizados por la CIE en 1.976 para lograr una representación perceptualmente uniforme del color. De esta manera, los colores se representan en el espacio a unas distancias proporcionales a las diferencias visuales entre ellos [19].

La variable L^* , es una medida de luminancia, mientras que las componentes a^* y b^* definen señales de color magenta-verde, y amarillo-cyan, respectivamente. Un valor negativo de a^* define un color más verde que magenta, mientras que un valor positivo de b^* define un color más amarillo que cyan [19].

El sistema CIELAB es utilizado en aquellas aplicaciones que requieran una medida precisa de la distancia perceptual entre dos colores, por ejemplo, en la comparación de resultados de filtrado de imágenes, la métrica de distancia ΔE ofrece una buena medida del efecto del

filtro sobre la imagen original. Otra ventaja del modelo CIELAB es que la señal de luminancia puede procesarse de forma individual sin verse afectada la cromaticidad de la imagen [19].

6.5.3. Modelo HSI

La familia de espacios HSI se derivan del modelo RGB a partir de una transformación de coordenadas. Con la transformación, el cubo RGB pasa a tener forma cilíndrica. Los modelos HSI poseen aspecto cilíndrico, de forma que la saturación se corresponde con un valor de distancia radial, mientras que el matiz es función de ángulo en el sistema de coordenadas polar. La intensidad es la distancia a lo largo del eje perpendicular al plano de coordenadas polares [19].

6.6. Detección de líneas

Algoritmo de Canny

El algoritmo de Canny se puede aplicar para la detección de bordes en una imagen. Para ello se realizan las siguientes fases [20]:

1. Primero se utiliza un filtro Gaussiano a partir de un *kernel* de 5x5, para evitar falsos positivos y eliminar el ruido.
2. Luego de la de tener la imagen suavizada, se aplica un *kernel* de Sobel en dirección horizontal y vertical para obtener la primera derivada en cada una de las direcciones. A partir de las imágenes, se puede obtener el gradiente del borde G y la dirección del píxel.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \quad (1)$$

$$G = \sqrt{G_x^2 + G_y^2}, \quad \theta = \arctan \frac{G_y}{G_x}. \quad (2)$$

3. La dirección que se obtiene del gradiente es siempre perpendicular a los bordes, redondeado a uno de los ángulos de las direcciones vertical, horizontal y las dos diagonales.
4. Después se realiza un escaneo completo de la imagen para eliminar los píxeles no deseados, verificando en cada píxel si se trata de un máximo local en vecindad en la dirección del gradiente.
5. Por último, los umbrales con mayor detección serán los que se tomarán como los bordes detectados, como se logra observar en la Figura [6].

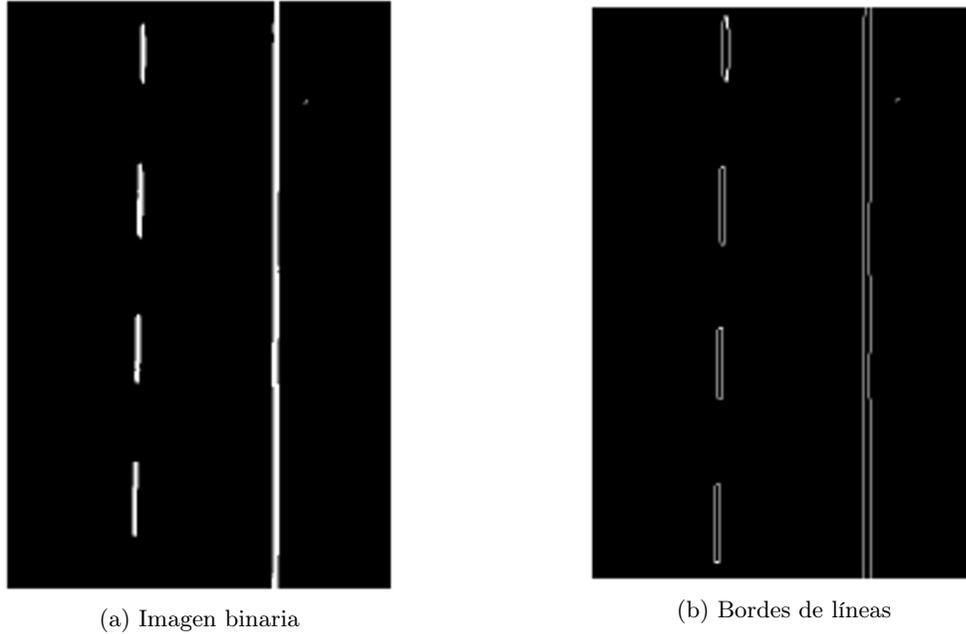


Figura 6: Ejemplo de detección de bordes en carriles utilizando el algoritmo de Canny [20](#).

6.6.1. Algoritmo de RANSAC en la detección de línea

El algoritmo RANSAC elige aleatoriamente pares de bordes detectados para formar una predicción de línea y luego prueba cuántos otros bordes caen en esta línea, en donde utiliza los siguientes pasos:

1. Seleccionar un número mínimo de puntos aleatoriamente del conjunto de datos. Estos puntos se llaman *inliers* potenciales.
2. Se ajustará un modelo a los puntos seleccionados aleatoriamente. En el caso de ajuste de línea, se puede utilizar la técnica de mínimos cuadrados para encontrar la mejor línea que se ajuste a los puntos.
3. Se calculará el número de *inliers* que se ajustan al modelo dentro de un umbral de distancia predeterminado. Los puntos que están cerca del modelo se consideran *inliers*.
4. Repetir los pasos 1-3 para un número específico de iteraciones.
5. Seleccionar el modelo que tenga la mayor cantidad de *inliers*.
6. Refinar el modelo final utilizando todos los *inliers* encontrados en el paso anterior. Esto se puede hacer mediante una estimación más precisa de los parámetros del modelo utilizando todos los puntos *inliers*, como se observa en la Figura [7](#) en donde detecta las líneas de los carriles de la carretera.

Los parámetros de entrada del algoritmo, como el número de iteraciones y el umbral de

distancia, deben ajustarse de acuerdo con el problema específico y al ruido presente en los datos [7].



Figura 7: Ejemplo de detección de líneas de carriles utilizando el algoritmo de RANSAC [7].

6.6.2. Transformada de Hough para detección de línea

La transformada de Hough permite detectar cualquier forma, cuando esta se pueda representar de manera matemática. La detección es robusta y consigue detectar las formas o líneas, aunque la imagen este dañada o tenga ruido. En el caso de la transformada de Hough, las líneas se expresan en el sistema polar de la siguiente manera:

$$y = \frac{\cos \theta}{\sin \theta} x + \frac{\rho}{\sin \theta}. \quad (3)$$

Para cada uno de los puntos iniciales (x, y) se puede definir una familia de rectas que pasa por dicho punto de la forma:

$$\rho_{\theta} = x_0 \cos \theta + y_0 \sin \theta, \quad \rho > 0, \quad 0 < \theta < 2\pi. \quad (4)$$

Se realiza el proceso para todos los puntos de la imagen y se buscan las intersecciones de las correspondientes familias de rectas. Dichas intersecciones darán pertenencia a la misma recta, mientras mayor cantidad de intersecciones se encuentren, la recta estará formada por más puntos. Para asegurar una detección robusta se define un umbral mínimo de intersecciones necesarias para detectar una línea. Un ejemplo de ellos se puede observar en la Figura [8] en donde al detectar las rectas de los carriles las coloca de color rojo [20].

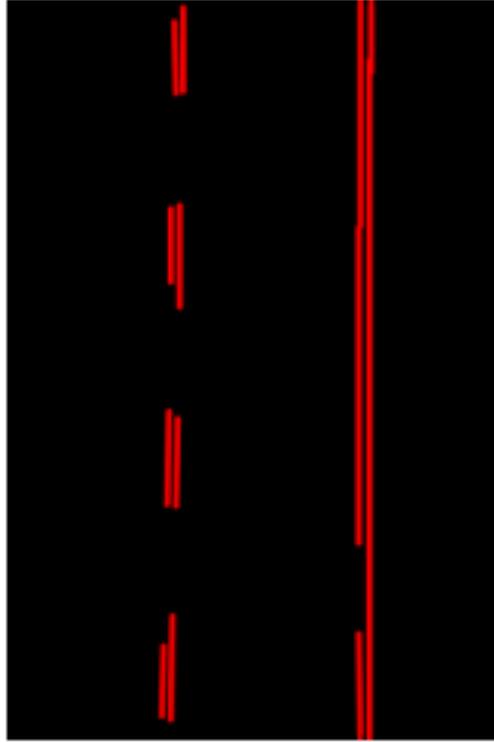


Figura 8: Ejemplo de detección de líneas de carriles utilizando la transformada de Hough [20].

6.6.3. Regresión Theil–Sen

El estimador Theil-Sen es robusto frente a los valores atípicos. Tiene un punto de ruptura de aproximadamente el 29.3 % en el caso de una regresión lineal simple, lo que significa que puede tolerar datos valores atípicos de hasta el 29.3 % en el caso bidimensional. La estimación del modelo se realiza calculando las pendientes y los interceptos de una subpoblación de todas las combinaciones posibles de p puntos de submuestra. Si se ajusta un intercepto, p debe ser mayor o igual que $n_features + 1$. La pendiente y el intercepto finales se definen entonces como la mediana espacial de estas pendientes e interceptos [21].

6.6.4. Algoritmo de clasificación de líneas

Luego de realizar un algoritmo de detección, se clasifican las líneas de los carriles en izquierdo, derecho o central. Para la clasificación se calculan 2 vectores \vec{V}_{cd} y \vec{V}_{cd} , los cuales tienen su origen en el centro de sus respectivos carriles y representan un ángulo respecto a la vertical con el ángulo de seguimiento θ_0 . Luego se toman los puntos iniciales y finales de la recta detectada y se calculan los vectores entre dichos puntos con los puntos centrales de los carriles derecho $(d_{n,i}, d_{n,f})$ e izquierdo $(i_{n,i}, i_{n,f})$. Para determinar la clasificación de la línea se calcula el producto vectorial de los vectores \vec{V}_{cd} y \vec{V}_{cd} por los vectores $d_{n,i}, d_{n,f}, i_{n,i}, i_{n,f}$.

$$\left. \begin{array}{l} \vec{V}_{cd} \times d_{1,i} \geq 0, \\ \vec{V}_{cd} \times d_{1,f} \geq 0, \end{array} \right\} \text{ Línea derecha,} \quad (5)$$

$$\left. \begin{array}{l} \vec{V}_{ci} \times \vec{d}_{1,i} \geq 0, \\ \vec{V}_{ci} \times \vec{d}_{1,f} \geq 0, \end{array} \right\} \text{Línea central,} \quad (6)$$

$$\left. \begin{array}{l} \vec{V}_{ci} \times \vec{d}_{1,i} < 0, \\ \vec{V}_{ci} \times \vec{d}_{1,f} < 0, \end{array} \right\} \text{Línea izquierda.} \quad (7)$$

Conociendo las medidas en píxeles de las líneas se realiza una conversión de para obtener la distancia real, conocer la inclinación de los carriles y poder realizar un rastreo[4].

6.7. Componentes de *hardware* del proyecto

6.8. OpenMV Cam H7

La OpenMV Cam H7 es una cámara especializada para aplicaciones de visión artificial, de tamaño pequeña (1.4 in x 1.75 in), con cámara integrada y un procesador ARM Cortex-M7 de alto rendimiento de 470 MHz con 1MB de SRAM y 2MB de memoria flash, lo que permite realizar procesamiento de imágenes en tiempo real y ejecutar algoritmos de visión por computador. Se puede utilizar programación de alto nivel con Python y cuenta con su propio IDE llamado OpenMV. Posee el sensor de imagen OV7725 el cual es capaz de capturar imágenes en escala de grises de 640x480 de 8 bits y en 16 bits RGB565 a 75 FPS y con una resolución de 320x240 a 150 FPS. El modelo de la cámara se puede observar en la Figura 9[22].



Figura 9: OpenMV Cam H7 de OpenMV[22].

6.8.1. Polulu 3pi+

El 3pi+ 32U4 OLED es un robot versátil y de alto rendimiento, programable por el usuario, que mide solo 9.6 cm de diámetro. En su núcleo se encuentra un microcontrolador AVR ATmega32U4 de Microchip (anteriormente Atmel), y al igual que los controladores programables A-Star 32U4, el 3pi+ 32U4 cuenta con una interfaz USB, cuenta con dos controladores de motor en puente H y una variedad de sensores integrados, incluyendo un par de codificadores cuadratura para el control de motor en lazo cerrado, una unidad de medición inercial completa (acelerómetro de 3 ejes, giroscopio y magnetómetro), cinco sensores de reflectancia orientados hacia abajo para seguir líneas o detección de bordes, y sensores de impacto izquierdo y derecho en la cara frontal del robot. Tres botones pulsadores integrados ofrecen una interfaz conveniente para la entrada del usuario, y una pantalla OLED gráfica de 128×64, un zumbador y LEDs indicadores permiten que el robot proporcione retroalimentación. En la Figura 10 se puede observar el diseño del Polulu 3pi(+)[23](#).

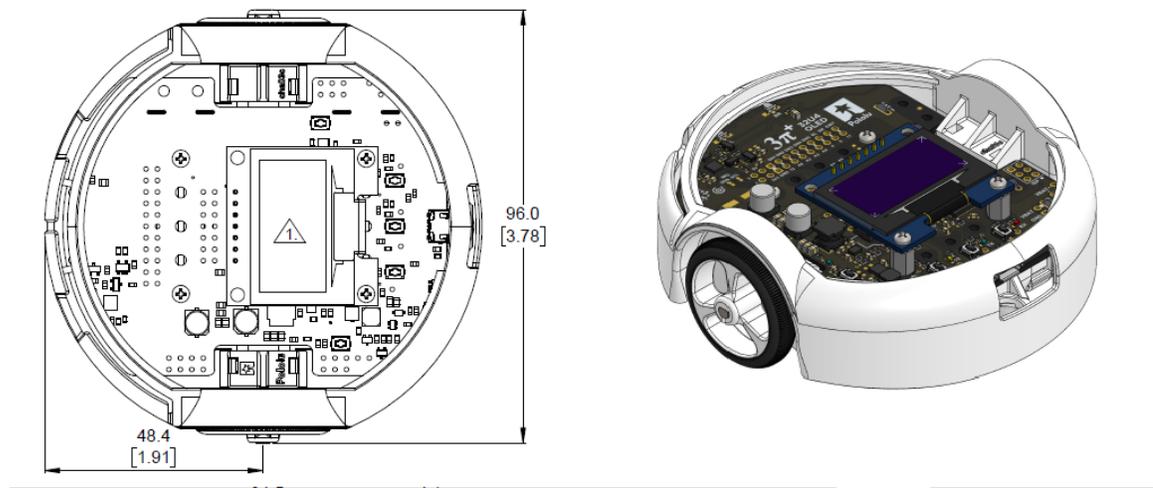


Figura 10: Diseño de Polulu 3pi+ con dimensiones[23](#).

6.8.2. Adafruit Neopixel

NeoPixel es la marca de Adafruit para píxeles y tiras de colores RGB individualmente direccionables basados en los LED/controladores WS2812, WS2811 y SK6812, utilizando un protocolo de control de *one wire*. Los LEDs rojos, verdes y azules están integrados junto con un chip controlador en un diminuto paquete de montaje en superficie que se controla a través de un solo cable. Pueden utilizarse de forma individual, encadenarse en cadenas más largas o ensamblarse en formas aún más interesantes. En la Figura 11 se puede observar un ejemplo de una matriz LED con Neopixel [24](#).

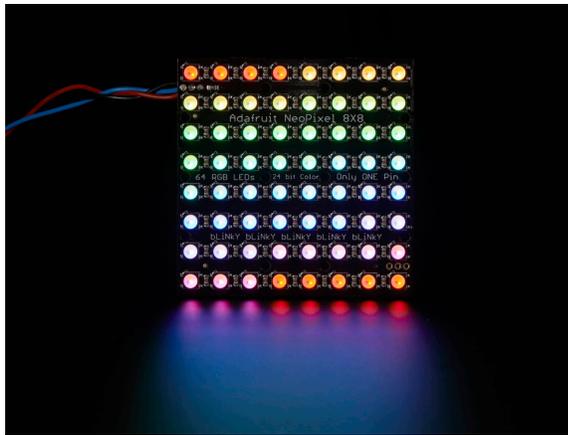


Figura 11: Matriz de 64 Neopixel [24](#).

Diseño y manufactura de infraestructura vial a escala

En el presente capítulo se detalla el diseño de la infraestructura vial a escala en la que se incluirán carreteras, señales de tránsito y semáforos. El ensamblaje de la infraestructura a escala debe de ser de manera simple y requerir el menor tiempo posible para el mismo.

7.1. Diseño de infraestructura de carriles

Para el diseño de la infraestructura se utilizó el software Autodesk Inventor 2023. Para realizar la escala, se emplearon las dimensiones de un carro Honda tipo sedán (1800 mm) y el modelo Pololu 3Pi+ (96 mm), obteniendo como resultado una escala de 18.75. Las dimensiones reales para el diseño de las carreteras a escala se obtuvieron del "Manual Centroamericano sobre señales viales uniformes" [25].

	Medida real(mm)	Medida escalado(mm)
Ancho de línea	150.00	8.00
Distancia entre líneas	3500.00	187.00
Distancia de segmentación de líneas	2350.00	125.00
Largo de línea segmentada	1875.00	100.00

Cuadro 1: Resultados de dimensiones escaladas para el diseño de carriles a escala.

7.1.1. Diseño de carriles rectos

Se realizaron carreteras de dos carriles, separadas por una línea discontinua, con la finalidad de poder simular carreteras de dos carriles con la misma dirección y carriles de una

vía con dirección contraria. Se evitó utilizar un patrón de unión entre carreteras con líneas rectas para prevenir posibles conflictos con la detección de líneas por visión de computadora. Se utilizó un patrón de rompecabezas, el patrón macho se colocó en la parte delantera y el patrón hembra en la parte trasera de la pieza. Además, se agregaron espacios en cada una de las esquinas para el ensamblaje de las piezas que soportan la estructura de los carriles, y se dejó espacio para poder colocar cables por debajo de la estructura de ser necesario. El diseño final de la carretera se puede observar en la Figura 12.

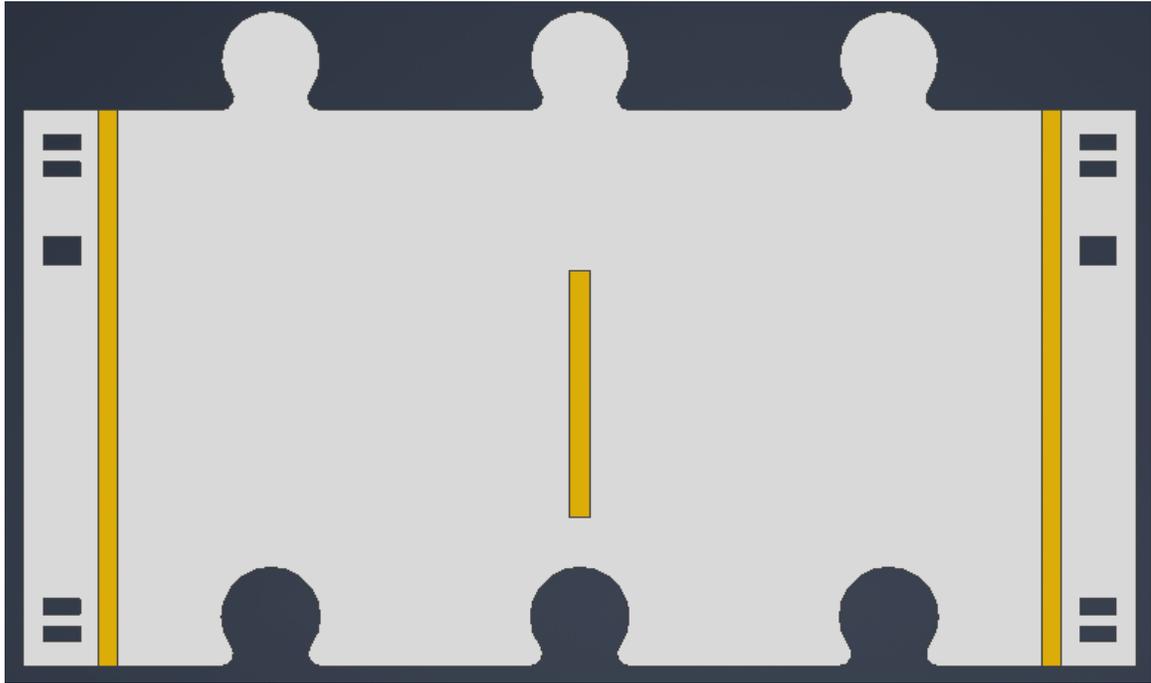


Figura 12: Diseño a escala de carretera de dos carriles, vista superior.

7.1.2. Diseño de carretera con esquina

Se realizaron carreteras de dos carriles con curva, las cuales cambian el sentido 90° de dirección, con la finalidad de poder simular curvas e intersecciones y evaluar el comportamiento de los algoritmos de detección de línea. Se conservó el mismo patrón de ensamblaje que las carreteras de dos carriles rectas, para que de igual manera el ensamblaje fuese compatible. En la Figura 13 se puede observar el primer diseño a escala de una carretera con esquina. El diseño de la esquina interior se modificó, se colocó el mismo el mismo radio que la esquina exterior, esto se realizó para mejorar la detección de la línea y facilitar el movimiento de la curva al controlador del vehículo a escala, el diseño se puede observar en la Figura 14.

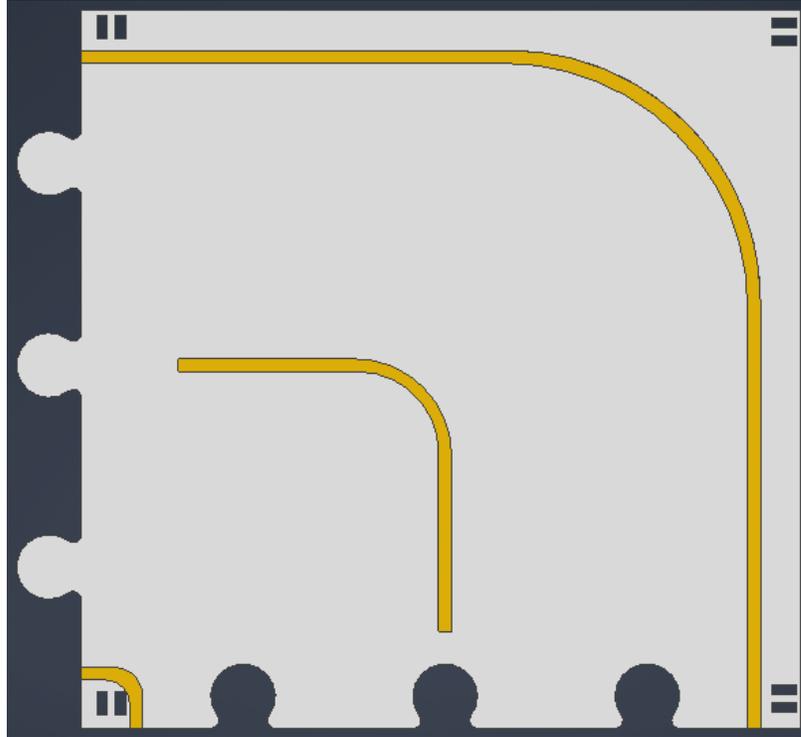


Figura 13: Diseño a escala de esquina de dos carriles primera versión, vista superior.

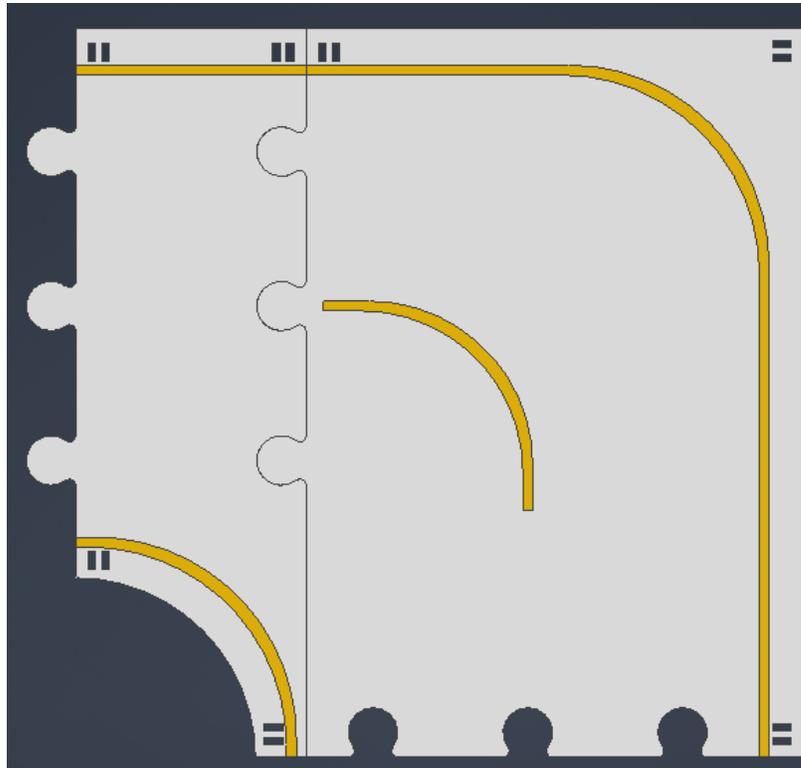


Figura 14: Diseño a escala de esquina de dos carriles segunda versión, vista superior.

7.1.3. Carretera con carril con intersección

Se diseñó carreteras con intersección a 90° en uno de los carriles, se utilizó el mismo radio de curva que en las carreteras con esquina. Se diseñó con la finalidad de controlar el flujo de tráfico en áreas donde convergen diferentes direcciones de circulación [25]. En esta intersección se puede separar el flujo del tráfico utilizando semáforos dobles. El diseño está dividido en tres piezas, esto para que se pueda acoplar en los diseños de las otras piezas diseñadas y cumplir con el tamaño máximo de la manufactura en la cortadora láser PLS4.5. El diseño se puede observar en la Figura [17].

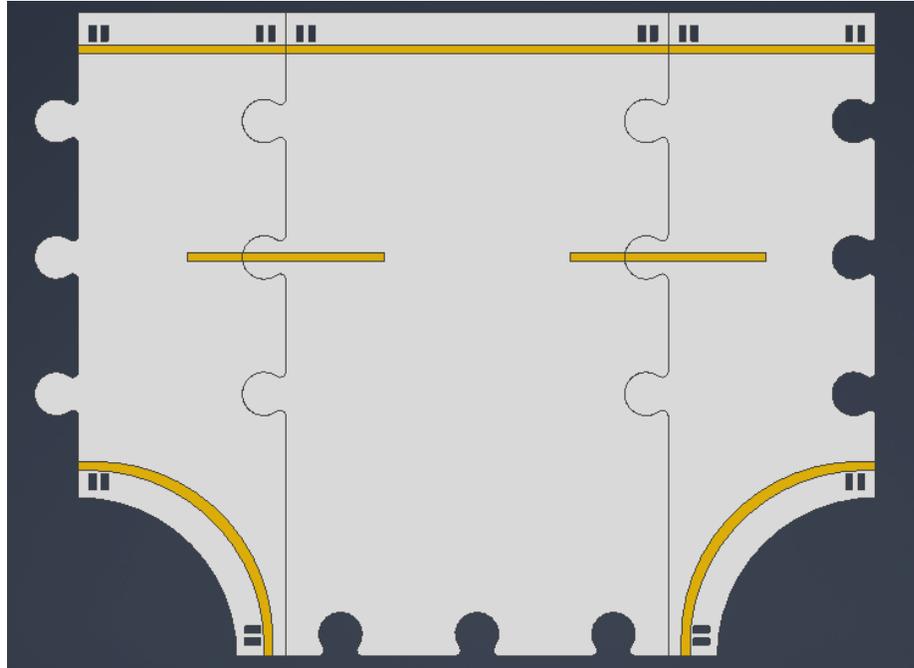


Figura 15: Diseño a escala de intersección de dos carriles, vista superior.

7.1.4. Carretera con intersección de cuatro carriles

Se diseñó carretera con intersección de cuatro carriles o cruces de cuatro esquinas, la finalidad es permitir que los vehículos crucen de una calle a otra en diferentes direcciones. Esto implica proporcionar señales, semáforos o reglas de prioridad para regular el flujo de tráfico y permitir que los vehículos se muevan de manera segura de un lado a otro [25]. El diseño está dividido en cuatro piezas, esto para que se pueda acoplar en los diseños de las otras piezas diseñadas y cumplir con el tamaño máximo de la manufactura en la cortadora láser PLS4.5. El diseño se puede observar en la Figura [16].

En la Figura [16] se puede observar el diseño del ensamblaje de un circuito en donde se utilizó las carreteras rectas, con esquina, con intersección e intersección de cuatro carriles.

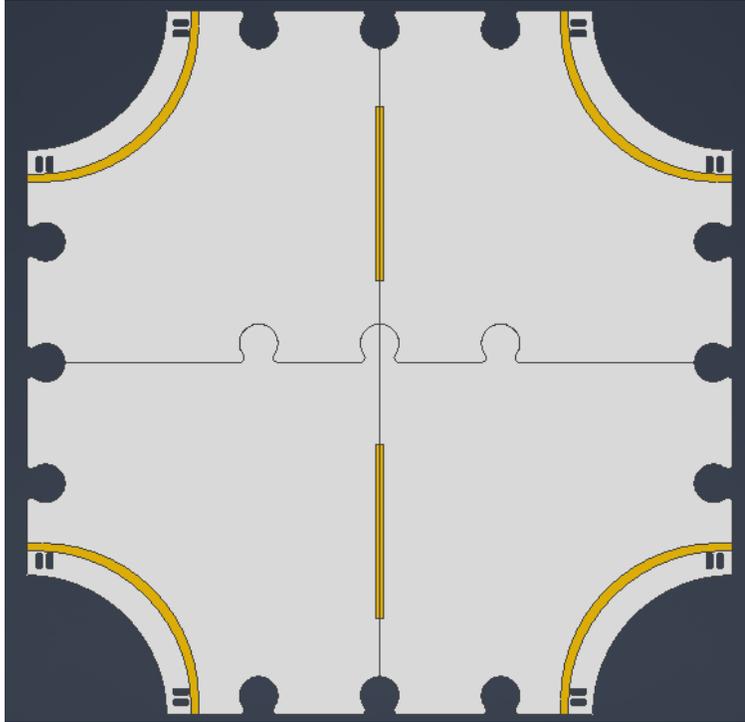


Figura 16: Diseño a escala de cruce de cuatro carriles, vista superior.

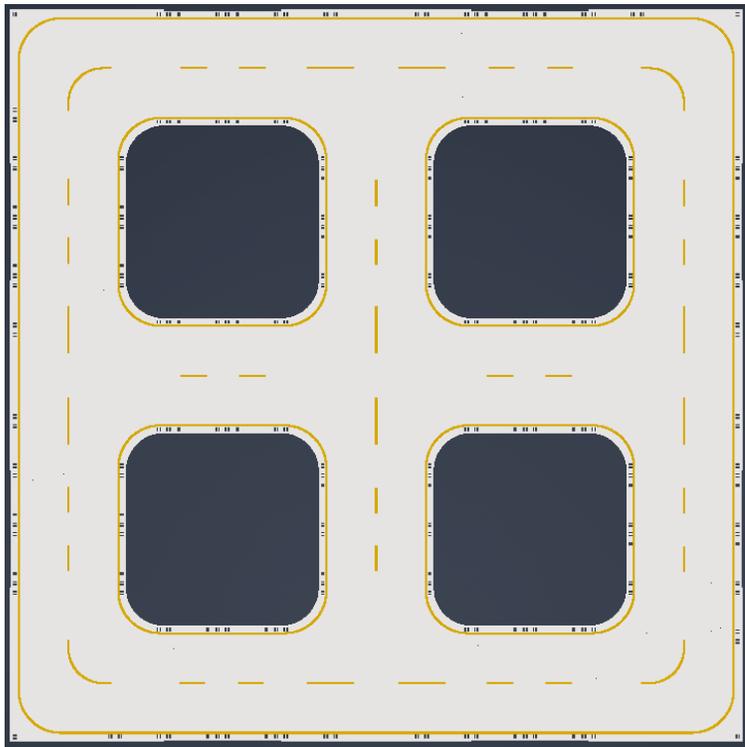


Figura 17: Diseño a escala del ensamble carreteras, esquinas, intersecciones y cruce de cuatro carriles, vista superior.

7.1.5. Acople de carretera

Se diseñó un acople para la carretera con el propósito de elevar la estructura de las carreteras y restringir su movimiento en la superficie. El diseño del acople se divide en tres partes. La parte inferior está diseñada para poder acoplar a presión dos ventosas de 4.0 cm, cada una en un extremo de la pieza, las cuales se adhieren a la superficie. La parte del centro está diseñada para acoplar cuatro piezas de forma de prisma rectangular. La parte superior tiene la longitud de la carretera, y se ensambla una parte del centro en cada costado de la pieza, esto con la finalidad de brindar soporte axial al patrón de rompecabezas. El diseño de la pieza ensamblada se puede observar en la Figura 18.

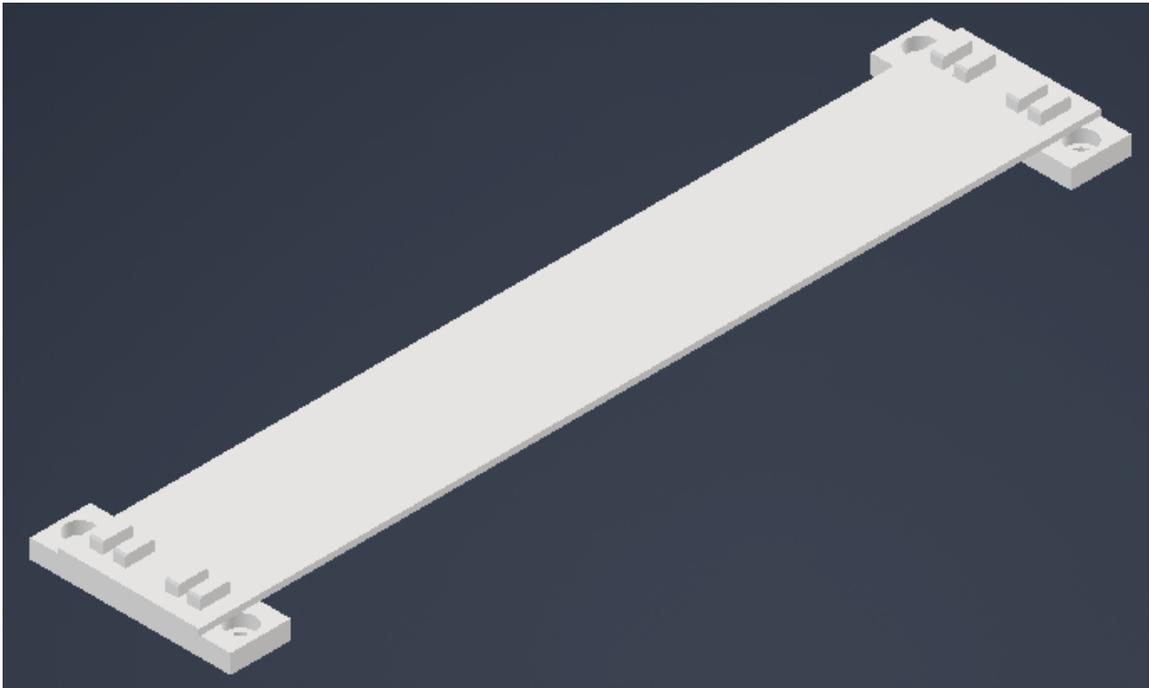


Figura 18: Diseño de acople de carretera a escala, vista isométrica.

7.1.6. Manufactura de carretera

Se escogió el material MDF de 3 mm de grosor y para la manufactura se utilizó la impresora láser PLS4.5. En este diseño se utilizó la técnica de corte para el contorno y agujeros del diseño de la carretera, además, se utilizó la técnica de grabado para marcar la línea de los carriles de las carreteras.

Se modificó el color del MDF de la carretera a color gris, para generar un contraste con la línea amarilla de los carriles y facilitar la detección de color de línea. Se utilizó pintura de laca con *thinner*, se utilizó cinta de aislar de color amarillo para adherir y colocarle el color a la línea de los carriles en las carreteras. En la Figura 19 se puede observar el diseño final de la carretera manufacturado y en la Figura 22 se puede el diseño final de la esquina manufacturado. En el Cuadro 2 se puede observar la cantidad piezas manufacturadas.

Pieza	Cantidad
Carril recto	132
Carretera con esquina	4
Acople de carretera	72

Cuadro 2: Cantidad de piezas manufacturadas a escala.

El acople manufacturado En la Figura 23 se puede observar el circuito de carretera y esquina armado en la plataforma del Robotat. Se utilizaron 40 carreteras a escala y 4 esquinas.



Figura 19: Carretera a escala manufacturada.

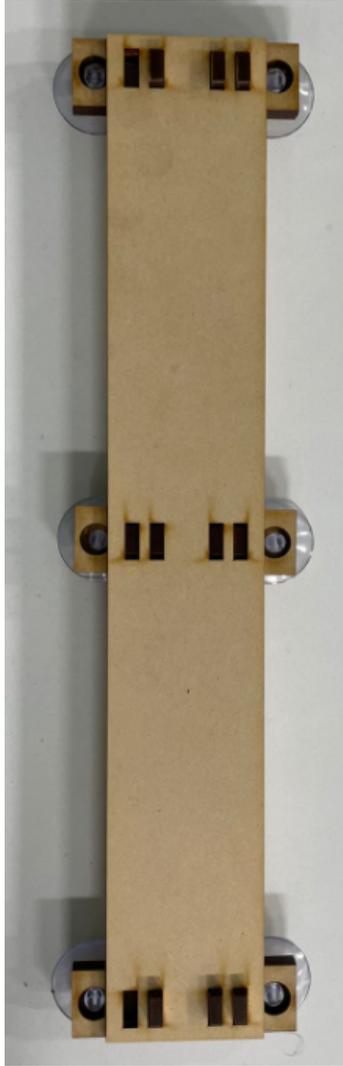


Figura 20: Acople de carretera manufacturada.

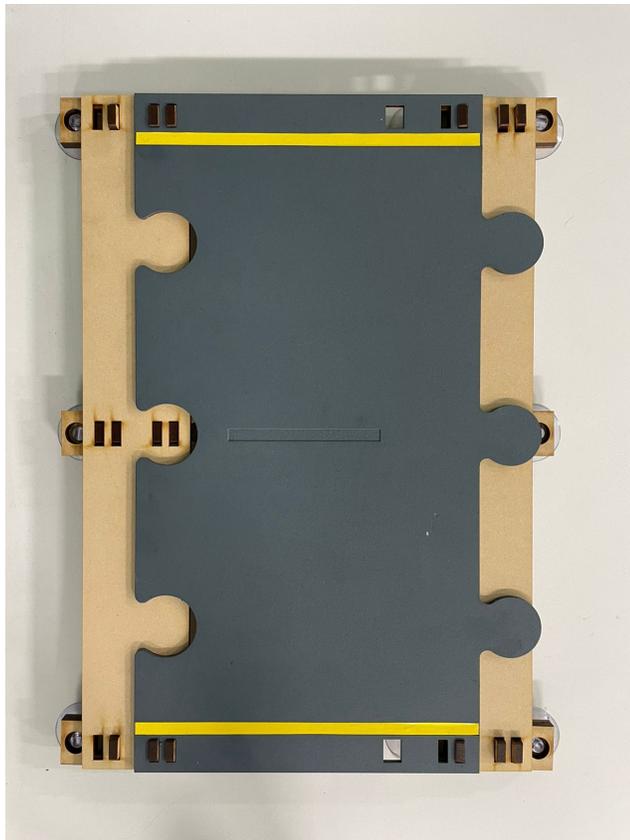


Figura 21: Acople y carretera a escala manufacturada.

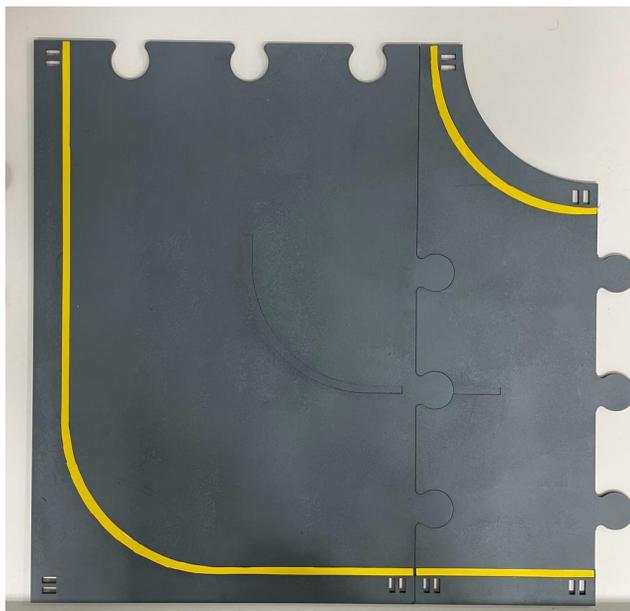


Figura 22: Esquina a escala manufacturada.

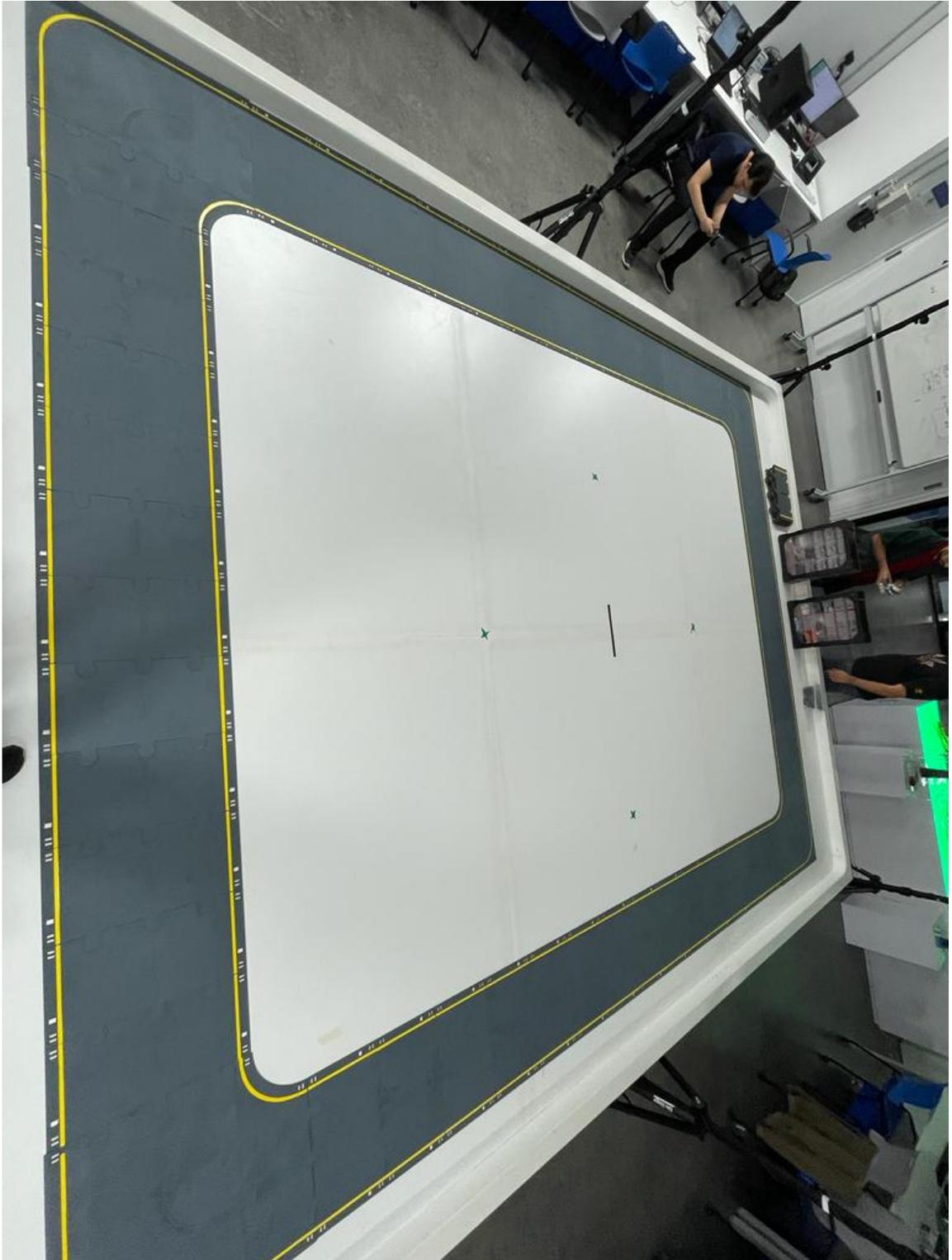


Figura 23: Circuito de carretera armado en la plataforma del Robotat.

7.2. Diseño de señales de tránsito

Se diseñaron tres señales de tránsito: forma octogonal para la señal de alto, forma de rombo para la señal de peatón y forma rectangular para las señales de límite de velocidad. Se utilizó la escala de 1:18.75 para el diseño de las señales en el software de Autodesk Inventor [Autodesk Inventor](#). Las dimensiones reales para el diseño de las señales de tránsito a escala se obtuvieron del "Manual Centroamericano sobre Señales Viales Uniformes" [25](#).

La altura libre de las señales de tránsito depende de la zona aplicada. En áreas urbanas, es de 2.10 m. Como se puede observar en la Figura [24](#), se utilizó la escala de 1:18.75 para el diseño de la altura de las señales de tránsito a escala. Para acoplar las señal de tránsito al diseño de carretera se realizó el diseño de prisma rectangular en la parte inferior de la pieza y se realizaron chaflanes en los esquinas para facilitar el acople de la pieza a la estructura [25](#).

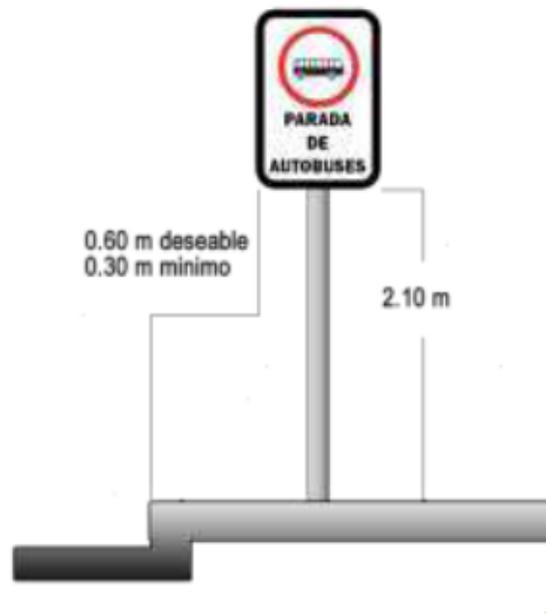


Figura 24: Dimensiones de altura de señales de tránsito en área urbana [25](#).

7.2.1. Señal de alto

La señal de alto se utiliza en los casos en donde el reglamento de tránsito indica al conductor que debe de detener por completo el vehículo antes de entrar a una calle o carretera principal, rampa, cruce o acera peatonal con prioridad de paso.

En la Figura [25](#) y el Cuadro [3](#) se puede observar el dimensionamiento de la señal de alto, en donde se utilizaron las dimensiones estándar para el diseño a escala. En la Figura [27a](#) se puede observar el diseño final [Autodesk Inventor](#): [25](#).



Figura 25: Diseño de dimensiones de señal de alto [25].

	Dimensiones (cm)					
	A	B	C	D	E	F
Ciclovía	45.7	1.0	18.2	13.4	15C*	1.5
Mínimo	61.0	1.6	24.0	17.9	20C*	2.0
Estándar	76.2	1.9	30.2	22.4	25C*	2.5
Especial	91.4	2.2	36.0	26.9	30C*	3.0

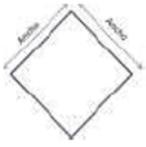
Cuadro 3: Tabla de dimensiones de la señal de alto [25].

7.2.2. Señal de peatón

La señal de peatón es una señal de prevención, la cual indica al conductor una advertencia de un peligro o cuidado en el entorno. Esta señal tiene de fondo el color amarillo. En el Cuadro 4 se puede observar las dimensiones utilizadas para el diseño a escala, en donde se utilizó el tamaño estándar. En la Figura 27c se puede observar el diseño final en Autodesk Inventor: [25].

7.2.3. Señal de límite de velocidad

La señal de límite de velocidad notifica a los conductores el valor de velocidad máxima a la cual deben de circular los vehículos. Esta señal tiene de fondo el color blanco y el valor de velocidad se encuentra dentro de la orla roja. En el Cuadro 4 se puede observar las dimensiones utilizadas para el diseño a escala, en donde se utilizó el tamaño estándar. En la Figura 27b se puede observar el diseño final en Autodesk Inventor: [25].

	Forma	Señales de Reglamentación			Señales de Prevención		
		Ancho (cm)	Alto (cm)	Figura	Lado A (cm)	Lado A (cm)	Figura
Mínimo	Rectángulo	46	71				
	Rombo				61	61	
Estándar	Rectángulo	61	91		76	76	
	Rombo						
Especial	Rectángulo	91	140		91	91	
	Rombo						

Cuadro 4: Tabla de dimensiones de señales de rectangulares y prevención [25].



(a) Señal de alto



(b) Señal de velocidad.



(c) Señal de peatón.

Figura 26: Diseños 3D finales de las señales de tránsito.

7.2.4. Manufactura de señales de tránsito

Se escogió el material MDF de 6 mm de grosor y para la manufactura se utilizó la impresora láser PLS4.5. En este diseño únicamente se utilizó la técnica de corte, por lo que en el diseño se colocaron los bordes del dibujo de la señal. Se realizó el diseño de la forma y los colores de cada una de las señales y se imprimió en papel calcomanía, para luego ser pegado en MDF luego de ser cortado. En la Figura [27] se puede observar el diseño final de las señales de tránsito.



(a) Señal de alto



(b) Señal de velocidad.



(c) Señal de peatón.

Figura 27: Diseños manufacturados finales de las señales de tránsito.

7.3. Diseño de semáforo

Los semáforos son dispositivos para el control del tránsito mediante los cuales se regula y ordena el movimiento de vehículos y peatones en calles y carreteras, a fin de que paren y procedan en forma alterna, por medio de luces de color rojo, amarillo y verde, operadas por una unidad de control. Los lentes de los semáforos para control de vehículos deben ser de forma circular. Existen dos diámetros nominales, de 0.20 m y de 0.30 m. Los diámetros de la parte visible de las lentes deben ser como mínimo de 0.197 m para las de 0.20 m y de 0.285 m para las de 0.30 m; los diámetros exteriores mínimos de los lentes serán de 0.213 m, para las de 0.20 m y de 0.305 m para las de 0.30 m, para realizar el diseño a escala se utilizó el diámetro nominal de 0.20 m, para la altura del poste del semáforo se utilizó el dimensionamiento que se puede observar en la Figura 28. Además, se agregaron espacios para colocar los cables como se puede observar en el diseño final de lá Figura 29 25.

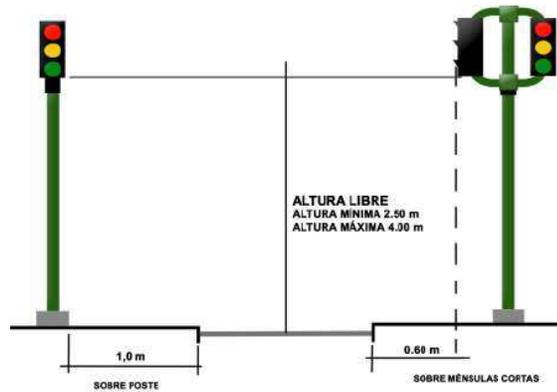


Figura 28: Diseño de dimensiones de altura de semáforo [25].



Figura 29: Diseño 3D de semáforo.

La interpretación de los colores de los semáforos es la siguiente:

- Verde fijo: Los conductores de los vehículos y el tránsito vehicular que observe esta luz podrá seguir de frente o girar a la derecha o a la izquierda
- Amarillo fijo: Los vehículos que enfrenten esta señal, deben detenerse antes de entrar al cruce, pues les advierte que el color rojo aparecerá a continuación.
- Rojo fijo: Los conductores de los vehículos deben detenerse antes de la línea de paso peatonal y si no existe, antes de la intersección y deben permanecer detenidos hasta que vean el verde correspondiente [25].

Se decidió utilizar los LEDs Neopixel para indicar el color de la luz de los semáforos por las siguientes razones: al utilizar el protocolo de comunicación *one wire*, es posible conectar múltiples LEDs utilizando un solo pin de comunicación en común. Los LEDs Neopixel se pueden programar para definir los colores y el nivel de luminosidad de manera individual, lo cual permite conectar múltiples semáforos y programarlo utilizando un microcontrolador. Se utilizó los conectores JST-SM de 3 pines para la conexión entre los semáforos, los cuales se pueden observar en la Figura 30.

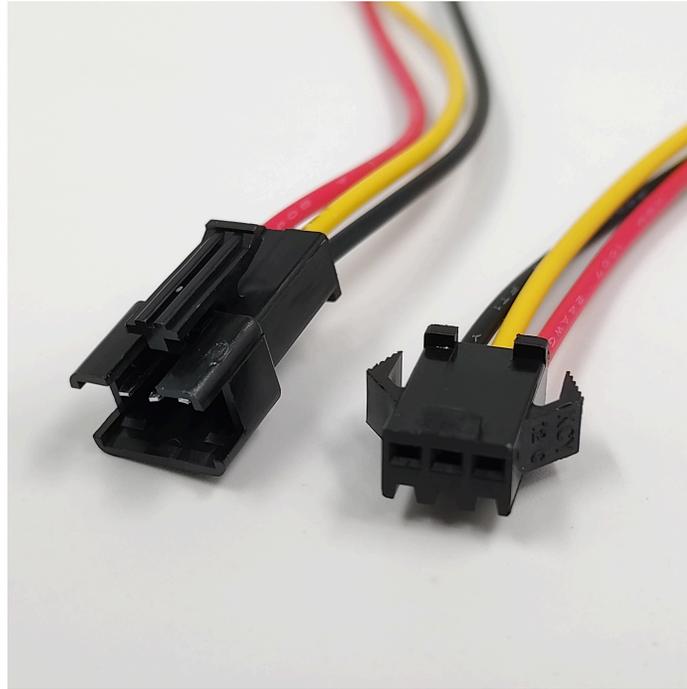
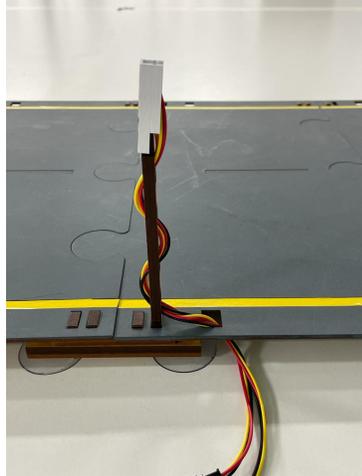


Figura 30: Conector JST-SM de 3 pines 26

En la Figura 31 se puede observar el diseño manufacturado del semáforo a escala, en la Figura 31a se puede observar el semáforo colocado a un costado de de la carretera, los conectores del semáforo se ingresaron por la ranura a un costado de la carretera a escala, en la Figura 31b se puede observar el semáforo a escala funcionando en estado verde.



(a) Vista lateral del semáforo manufacturado



(b) Semáforo funcionando en la carretera

Figura 31: Diseños manufacturados final del semáforo a escala.

7.3.1. Programación de los semáforos

Se utilizó la placa de desarrollo Arduino Nano para controlar los semáforos a escala por su tamaño compacto y necesitar un pin con salida `PWM:` además, se utilizó la librería de Neopixel [27] para controlar los colores y la intensidad de luz de cada Neopixel. Se colocó una intensidad del 15 % en los Neopixel [27], esto se colocó con la finalidad de facilitar a la OpenMV Cam H7 la detección la detección de color con luz.

Se realizó programación orientada objetos, se crearon 2 clases, la primer clase se llama semáforo y es de un semáforo simple, se colocó como atributos las direcciones y los tiempos de encendido de los tres estados del semáforo. Se puede observar el pseudocódigo [1] de la función de control de la clase semáforo. La segunda clase se llama semáforo doble, se colocó los atributos de las direcciones de los tres estados de los dos semáforos y los tiempos de encendido de las luces verde y amarilla de ambos semáforos. Se puede observar el pseudocódigo [2] de la función de control de la clase semáforo doble.

Algorithm 1 Función semáforo

Entrada: Tiempo en milisegundos del reloj

Salida:

```
1: switch Estado de semaforo do
2:   case Luz verde encendida
3:     Luz verde encendida
4:     if  $Tiempodelreloj \geq Tiempo\ encendido\ luz\ verde$  then
5:        $Tiempo\ actual = Tiempo\ del\ reloj$ 
6:        $Estado\ de\ semaforo = Luz\ amarilla\ encendida$ 
7:   case Luz amarilla encendida
8:     Luz amarilla encendida
9:     if  $Tiempodelreloj \geq Tiempo\ encendido\ luz\ amarilla$  then
10:       $Tiempo\ actual = Tiempo\ del\ reloj$ 
11:       $Estado\ de\ semaforo = Luz\ roja\ encendida$ 
12:   case Luz roja encendida
13:     Luz roja encendida
14:     if  $Tiempodelreloj \geq Tiempo\ encendido\ luz\ roja$  then
15:        $Tiempo\ actual = Tiempo\ del\ reloj$ 
16:        $Estado\ de\ semaforo = Luz\ verde\ encendida$ 
```

Algorithm 2 Función semáforo doble

Entrada: Tiempo en milisegundos del reloj

Salida:

```
1: switch Estado de semaforo do
2:   case Semaforo 1 luz verde encendida
3:     Luz verde encendida semáforo 1
4:     Luz roja encendida semáforo 2
5:     if  $Tiempodelreloj \geq Tiempo\ encendido\ luz\ verde$  then
6:        $Tiempo\ actual = Tiempo\ del\ reloj$ 
7:        $Estado\ de\ semaforo = Semaforo\ 1\ luz\ amarilla\ encendida$ 
8:   case Semaforo 1 luz amarilla encendida
9:     Luz amarilla encendida semáforo 1
10:    Luz roja encendida semáforo 2
11:    if  $Tiempodelreloj \geq Tiempo\ encendido\ luz\ amarilla$  then
12:       $Tiempo\ actual = Tiempo\ del\ reloj$ 
13:       $Estado\ de\ semaforo = Semaforo\ 2\ luz\ verde\ encendida$ 
14:   case Semaforo 2 luz verde encendida
15:     Luz roja encendida semáforo 1
16:     Luz verde encendida semáforo 2
17:     if  $Tiempodelreloj \geq Tiempo\ encendido\ luz\ verde$  then
18:        $Tiempo\ actual = Tiempo\ del\ reloj$ 
19:        $Estado\ de\ semaforo = Semaforo\ 2\ luz\ amarilla\ encendida$ 
20:   case Semaforo 2 luz amarilla encendida
21:     Luz roja encendida semáforo 1
22:     Luz amarilla encendida semáforo 2
23:     if  $Tiempodelreloj \geq Tiempo\ encendido\ luz\ amarilla$  then
24:        $Tiempo\ actual = Tiempo\ del\ reloj$ 
25:        $Estado\ de\ semaforo = Semaforo\ 1\ luz\ verde\ encendida$ 
```

Detección de infraestructura vial mediante visión de computadora

En este capítulo se explicará el desarrollo de la detección y clasificación de señales de tránsito y el estado de los semáforos. Se utilizaron dos métodos de clasificación de objetos, *Haar Cascade* y redes neuronales convolucionales en *TFLite (TensorFlow Lite)*. Además, se explican los conceptos y el desarrollo del modelo de detección de líneas para los carriles de la carretera, el cual será utilizado para la conducción autónoma de un vehículo dentro de la misma.

8.1. Detección de carriles mediante visión de computadora

Se inició colocando las configuraciones para la OpenMV Cam H7, donde se seleccionó el formato de color RGB565 para poder detectar el color de la línea del carril, así como un tamaño de resolución de 120x160 píxeles (QQVGA). Además, se volteó verticalmente y se reflejó horizontalmente la cámara, como se puede observar en el código [8.1](#).

```
1 sensor.reset()
2 sensor.set_pixformat(sensor.RGB565)
3 sensor.set_framesize(QQVGA)
4 sensor.set_vflip(True)
5 sensor.set_hmirror(True)
6 sensor.skip_frames(time = 200)
7 clock = time.clock()
```

Código 8.1: Parámetros iniciales de configuración de cámara.

Luego se realizó una regresión en la que se seleccionó el color amarillo y se establecieron los valores mínimos y máximos para los canales LAB L, A y B, respectivamente. Se utilizó la regresión lineal de Theil-Sen al configurar *robust = True*, la cual calcula la mediana de todas las pendientes entre los píxeles que han superado el umbral en la imagen, como se

puede observar en el Código 8.2.

```

1 line = img.get_regression([[50, 100, -128, 127, -128, 127]], \
2     area_threshold = AREA_THRESHOLD,
3     pixels_threshold = PIXELS_THRESHOLD, \
4     robust = True)

```

Código 8.2: Detección de línea amarilla.

Si la magnitud del valor de la línea detectada anteriormente es mayor al *threshold*, se obtiene la posición horizontal de la línea detectada utilizando la transformada de Hough de la Ecuación 8. Luego se calculó la distancia desde el centro de la posición horizontal de la línea. Por último se normalizó la distancia dividiéndola por la mitad del ancho de la imagen, esto se puede observar en el Código 8.3.

$$x = \frac{\rho - y \sin \theta}{\cos \theta}. \quad (8)$$

```

1 cy = img.height() / 2
2 cx = (line.rho() - (cy * math.sin(math.radians(line.theta()))))
3     /math.cos(math.radians(line.theta()))
4
5 cx_middle = cx - (img.width() / 2)
6 cx_normal = cx_middle / (img.width() / 2)

```

Código 8.3: Posición de línea amarilla detectada y normalización.

El resultado de aplicar el algoritmo de detección de línea en la carretera a escala se puede observar en la Figura 32 y 33, en donde se colocó una línea roja cuando se detecta una línea de color amarillo. La línea roja trata de alinearse a una línea vertical.

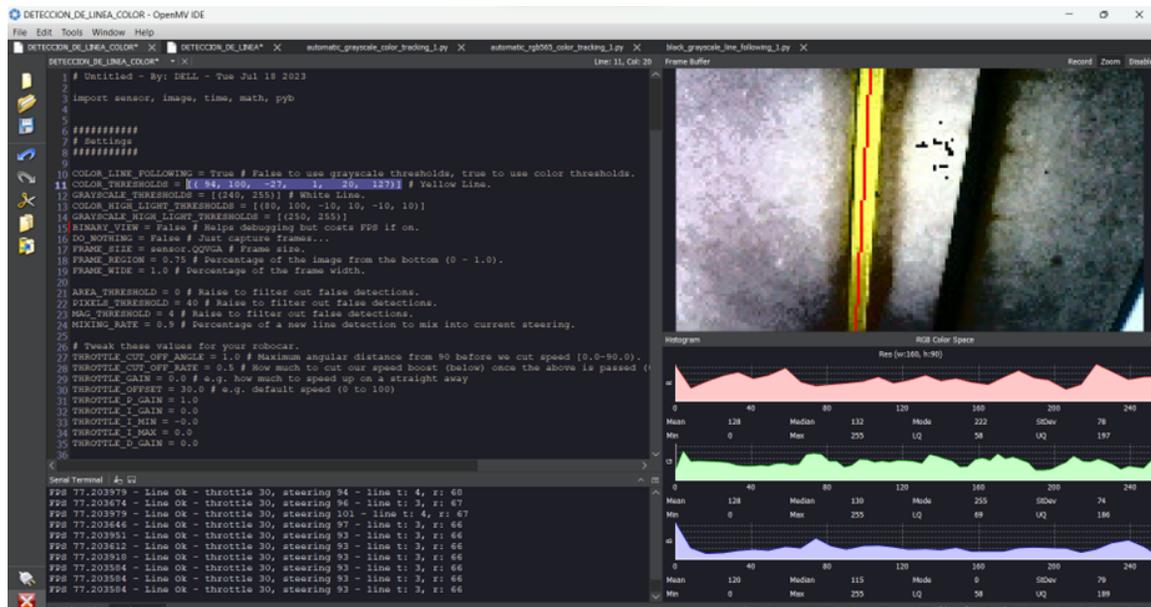


Figura 32: Resultado de aplicación de algoritmo de detección de línea en carretera a escala en el IDE de OpenMV.



Figura 33: Resultado de aplicación de algoritmo de detección de línea en carretera a escala.

8.2. Haar cascade

Cuando se realiza un entrenamiento *cascade* se genera un archivo *.xml*, este archivo se debe convertir a un archivo *.cascade* para que pueda ser utilizado en la OpenMV Cam H7. Para hacerlo, es necesario tener la versión de Python 2.7 en Linux. Luego, se ejecuta el comando *cascade_convert.py* seguido del nombre del archivo *.xml* en la terminal de Linux, el ejemplo de como ejecutar el comando se puede observar en el Código 8.4 [14].

```
1 ~ python 2.7 cascade_convert.py archivo.xml
```

Código 8.4: Comando para conversión de archivo *.xml* a *.cascade*.

8.2.1. Detección de señal de alto

Se utilizó el archivo *stop_sign_classifier_2.xml* realizado por Mauricio Reyes [28], el cual clasifica señales de alto en el área urbana. Se convirtió a un archivo *cascade* con el comando 8.4. El archivo se carga en la memoria de la OpenMV Cam H7 para que luego pueda ser utilizado.

Se colocó las configuraciones iniciales para la OpenMV Cam H7

```
1 sensor.reset()
2 sensor.set_pixformat(sensor.RGB565)
3 sensor.set_framesize(QQVGA)
4 sensor.set_vflip(True)
5 sensor.set_hmirror(True)
6 sensor.skip_frames(time = 200)
7 clock = time.clock()
```

Código 8.5: Parámetros iniciales de configuración de cámara.

Se colocó el nombre del archivo al *feature descriptor*, el cual permite la identificación y el seguimiento de objetos y patrones en imágenes y vídeos, y el número de etapas (*stages*), un número menor realiza el procesamiento de manera más rápida, pero se tiene el costo de detectar falso positivos. En el siguiente Código 8.6 se puede observar la configuración del clasificador de señal de alto.

```
1 stop_cascade = image.HaarCascade("stop.cascade", stages=25)
```

Código 8.6: Método de detección de características Haar cascade.

En el bucle se buscarán siempre las características de la señal de alto. Se puede establecer un umbral de detección. Mientras el umbral se encuentre más cercano a 1.0, disminuye el número de detecciones, pero la detección de falsos positivos es menor. Luego, si detectaba un objeto se colocó un rectángulo alrededor de la señal de alto detectada, como se puede observar en el Código 8.7.

```
1 objects = img.find_features(stop_cascade,
2 threshold=0.9, scale_factor=1.25)
3
4 for r in objects:
5     img.draw_rectangle(r)
```

Código 8.7: Detección de de señal de alto.

En la Figura 34 se puede observar la aplicación del algoritmo de detección de la señal de alto creada a escala. Puede observarse que al momento de detectar, se crea un cuadro de color blanco alrededor de la señal.



Figura 34: Resultado de aplicación de algoritmo de detección de señal de alto con *Haar cascade*.

8.2.2. Detección de vehículos

Se utilizó el archivo *cars.xml* realizado por Andrews Cordolino [29], el cual clasifica vehículos. Se convirtió a un archivo *cascade* con el comando [8.4]. El archivo se carga en

la memoria de la OpenMV Cam H7 para que luego pueda ser utilizado. Se utilizaron las configuraciones utilizadas en la detección de señal de alto. Se colocó una imagen de un vehículo, en la Figura 35 se puede observar la detección positiva del vehículo.



Figura 35: Resultado de aplicación de algoritmo de detección vehículo con *Haar cascade*.

8.3. *TensorFlow Lite*

8.3.1. *Edge Impulse*

Los pasos para crear un proyecto en *Edge Impulse* son los siguientes:

1. Crear una cuenta en la plataforma de *Edge Impulse*. Se puede crear una cuenta gratuita, la cual fue suficiente para realizar este proyecto.
2. Crear un nuevo proyecto, se colocó el nombre del proyecto y el tipo, en este caso se seleccionó *Developer*.
3. Luego se creó una *dataset*, en este caso las imágenes de señales de alto, peatón, y estados de semáforo se extrajeron de Internet, capturas [Google street view](#) de Guatemala y fotografías tomadas en la ciudad capital. Luego, se seleccionó la sección de *data acquisition* (cuadro color anaranjado en la Figura 36) y se seleccionó *upload data* (cuadro amarillo en la Figura 36), se escoge la carpeta en donde se encuentre la *dataset* y se escogió la opción *Automatically split between training and testing*, para separar el *dataset* en 80 % entrenamiento y 20 % testeo.

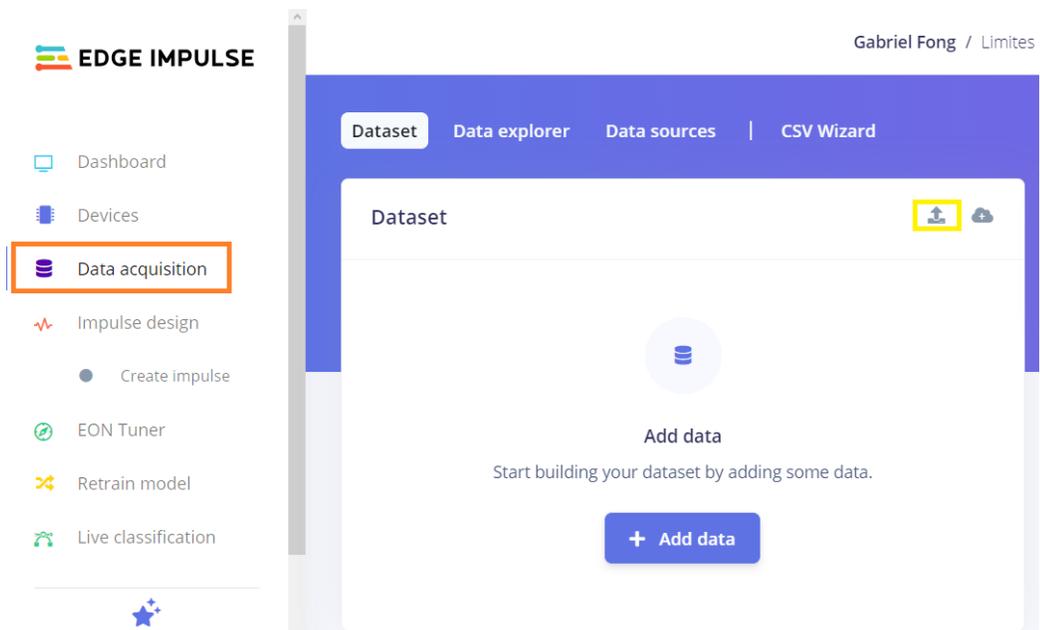


Figura 36: Sección *data acquisition* en la plataforma de *Edge impulse*.

4. A cada una de las imágenes del *dataset* de entrenamiento se colocó la o las etiquetas, se puede colocar múltiples etiquetas por imagen y pueden ser de diferentes tipos, se realizaron 5 clases: Semáforo en verde (SEV), semáforo en amarillo (SEA), semáforo en rojo (SER), señal de alto (ALTO) y señal de peatón (PEATON), el ejemplo del etiquetado se puede observar en la Figura 37.

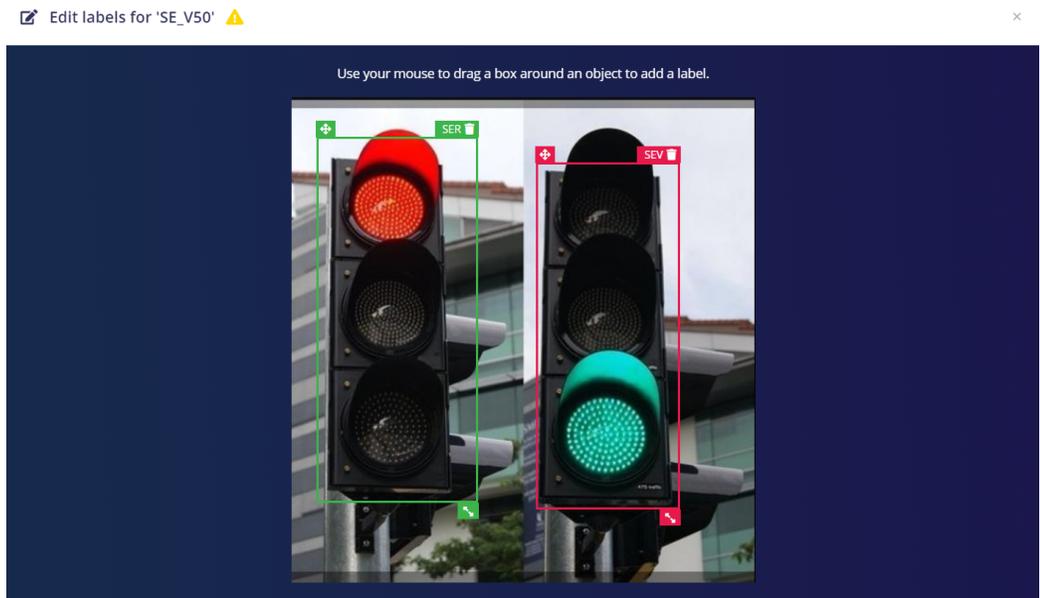


Figura 37: Etiquetado de semáforo en verde y rojo en la plataforma de *Edge impulse*.

Se obtuvo un total de 367 objetos etiquetados, en donde 65 fueron de señales de alto, 73 señales de peatón, 80 de semáforos en verde, 59 de semáforos en amarillo y 86 de

semáforos en rojo. Se utilizaron 294 etiquetados para el entrenamiento y 74 etiquetados para el testeo. Los resultados se pueden observar en la Figura 38.



Figura 38: Resultados de etiquetado de *dataset* en la plataforma de *Edge impulse*.

5. En la sección de *impulse design* y en *create impulse* se seleccionaron los bloques que se encuentran en la Figura 39, en la primera configuración que se realizó se utilizó un tamaño de imagen de 128x128 píxeles de reconocimiento, en el segundo entrenamiento se utilizó un tamaño de 96x96 píxeles de reconocimiento. En la salida se colocó las cinco clases mencionadas anteriormente.

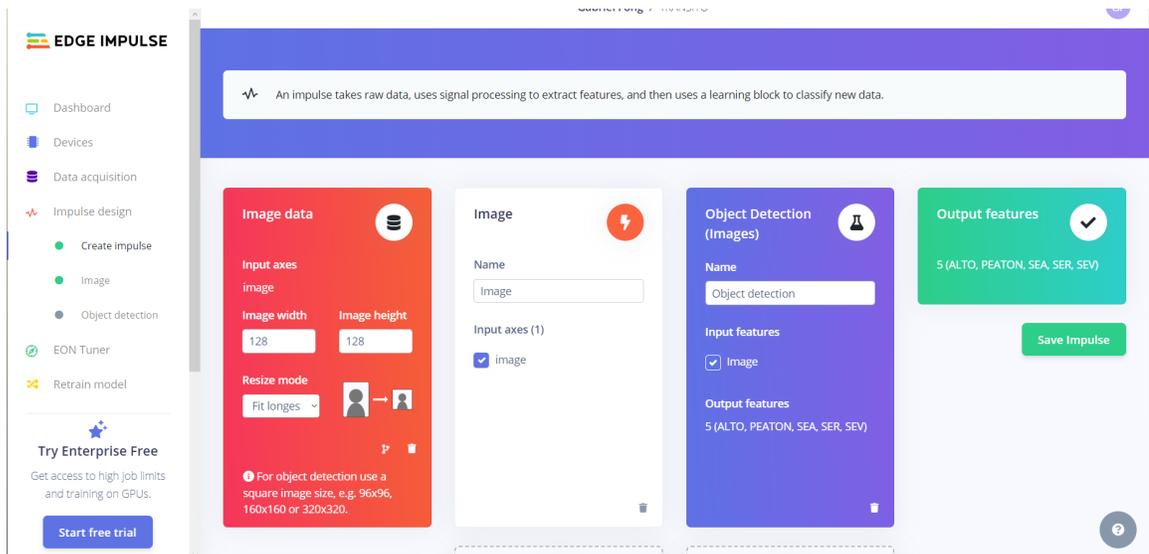


Figura 39: Configuraciones de bloques para el proceso de entrenamiento del modelo en la plataforma de *Edge impulse*.

6. Luego, en la en la sección de *impulse design* y en *object detection* se colocó las configuraciones de la red neuronal con 60 ciclos de entrenamiento, 0.001 de *validation rate*, 20% de *validation set size*, se utilizó el modelo de FOMO MobileNet V2 0.35 como arquitectura de red neuronal. La plataforma utiliza *Transfer learning*, en donde utiliza las características de un modelo pre entrenado para facilitar el entrenamiento de un nuevo modelo. *Edge impulse* apoya oficialmente este modelo de *Transfer learning* y garantiza que la detección se puede dar en escala de grises y la escala de color RGB. La configuración se puede observar en la Figura 40.

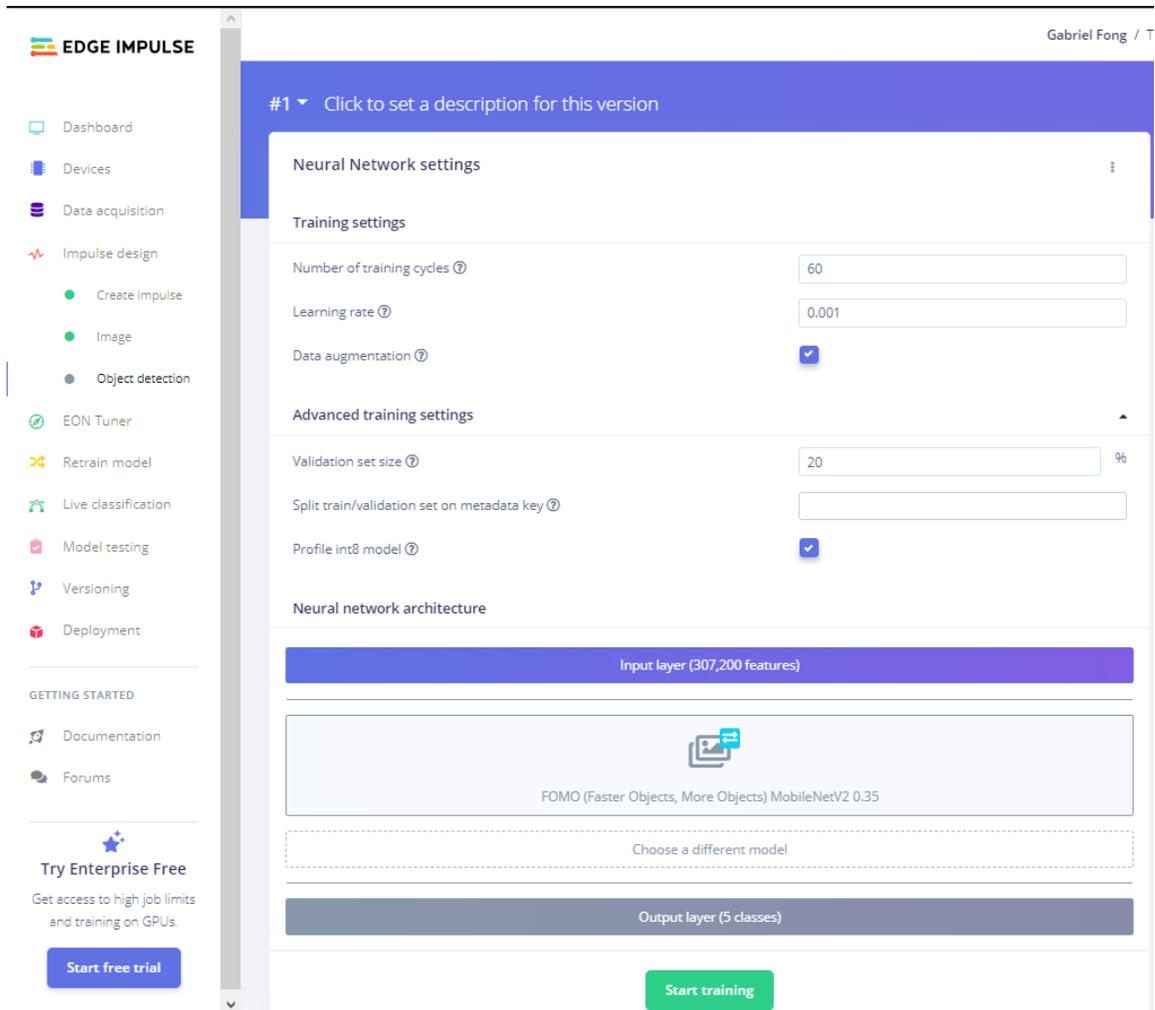


Figura 40: Configuraciones de red neuronal para el proceso de entrenamiento del modelo en la plataforma de *Edge impulse*.

7. Por último, en la sección de *deployment*, se seleccionó la opción de exportar a OpenMV como se observa en la Figura 41. Se generó 3 archivos: Un archivo de código para el IDE de OpenMV, un archivo .txt con los nombres de las clases colocadas y un archivo .tflite, el cual es el resultado del entrenamiento. Estos archivos se pueden observar en la Figura 42.

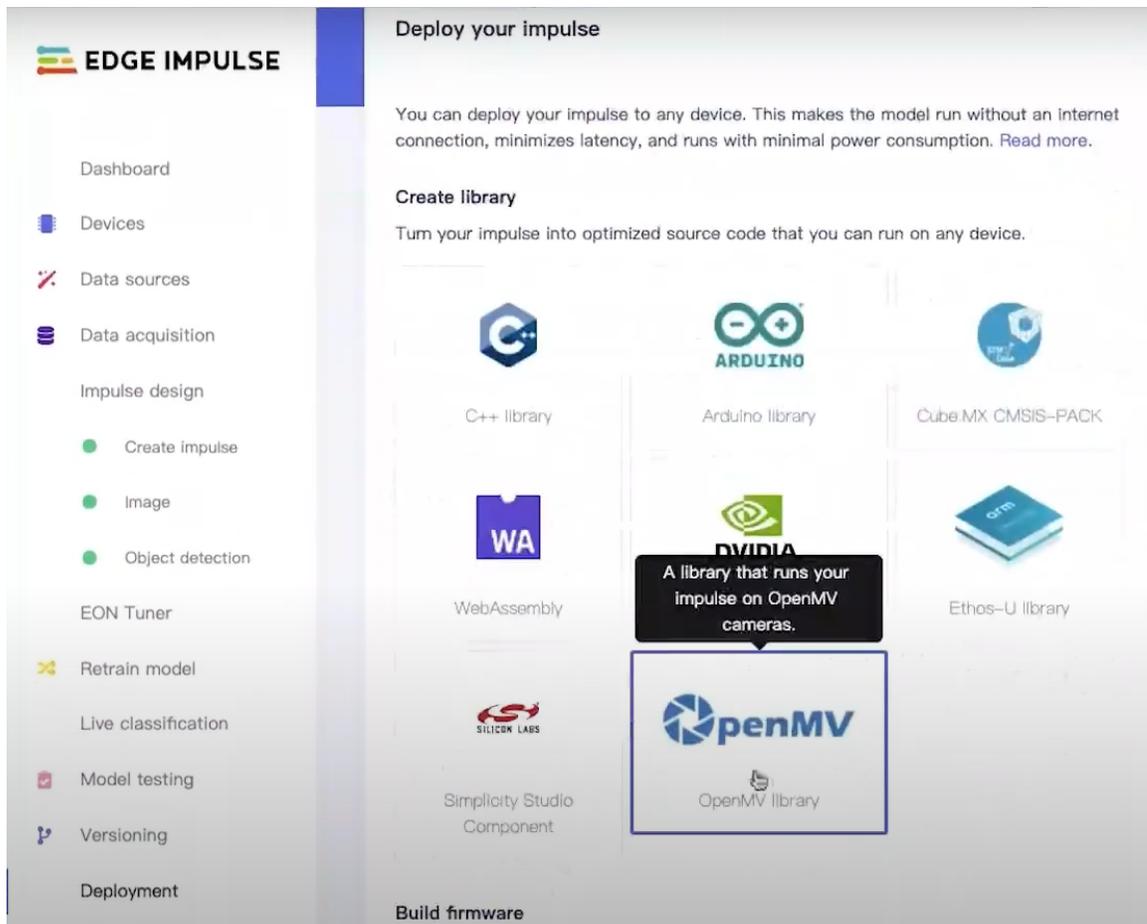


Figura 41: Selección de librería para exportar el modelo en la plataforma de *Edge impulse*.

<input type="checkbox"/> Name	Date modified	Type	Size
ei_object_detection.py	8/26/2023 11:59 AM	PyCharm2023.2	3 KB
labels	8/21/2023 11:06 PM	Text Document	1 KB
trained.tflite	8/21/2023 11:06 PM	TFLITE File	56 KB

Figura 42: Archivos exportados del modelo de entrenamiento.

8.3.2. Requisitos para utilizar *TensorFlow Lite* en OpenMV

1. Primero, se conectó la OpenMV Cam H7 a la computadora por USB.
2. Luego, copiar y pegar los archivos .txt y .tflite en la memoria de la OpenMV Cam H7.
3. Después, en la sección de *deployment* en el apartado de *Build firmware*, se selecciona la opción de OpenMV, como se observa en la Figura [43](#).

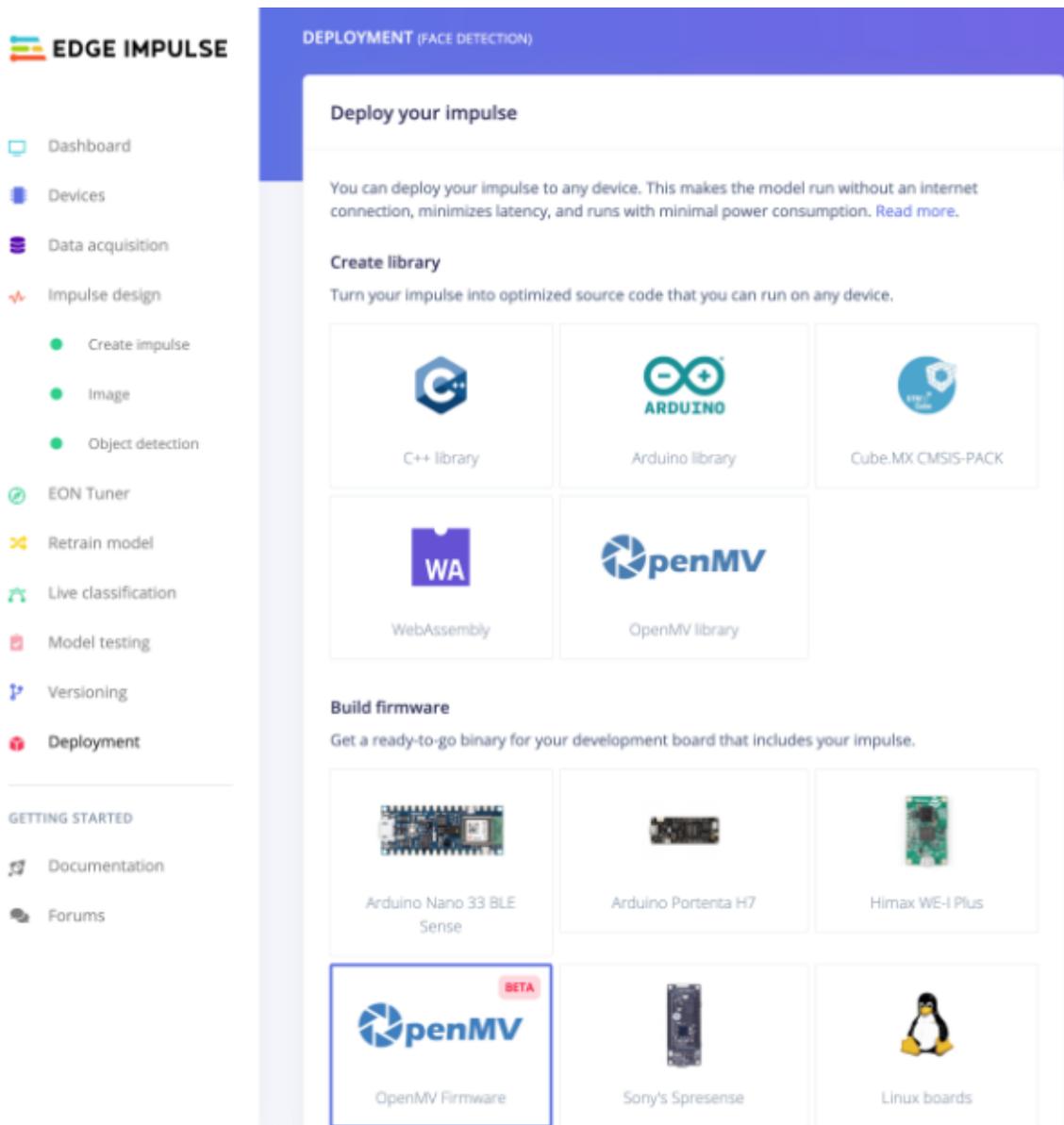


Figura 43: Selección de *firmware* a utilizar en la plataforma de *Edge impulse*.

4. Se guarda y descomprime el archivo .ZIP generado.
5. Se desconecta y se vuelve a conectar la OpenMV Cam H7 en la computadora.
6. Se presiona dos veces en botón de *reset* del dispositivo.
7. Luego, en el IDE de OpenMV se dirige a **Tools->Run Bootloader (Load Firmware)**, como se observa en la Figura 44

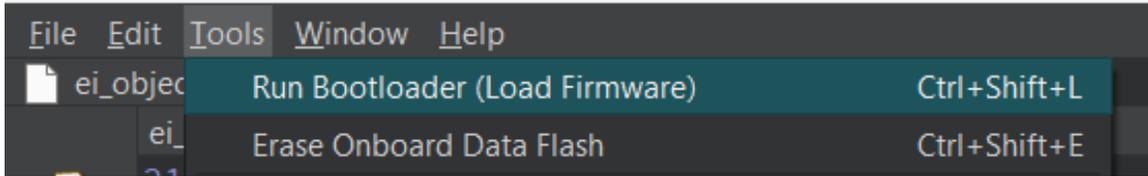


Figura 44: Selección del *firmware* a utilizar en el IDE de OpenMV.

- Después, se ingresa la ruta del archivo `edge_impulse_firmware_openmv_cam_h7.bin`, el cual es el que se necesita para el modelo de OpenMV Cam H7, se selecciona la opción de **Erase internal file system**, y se presiona **run**, como se puede observar en la Figura 45. En el dispositivo deberá de parpadear la LED en color azul.

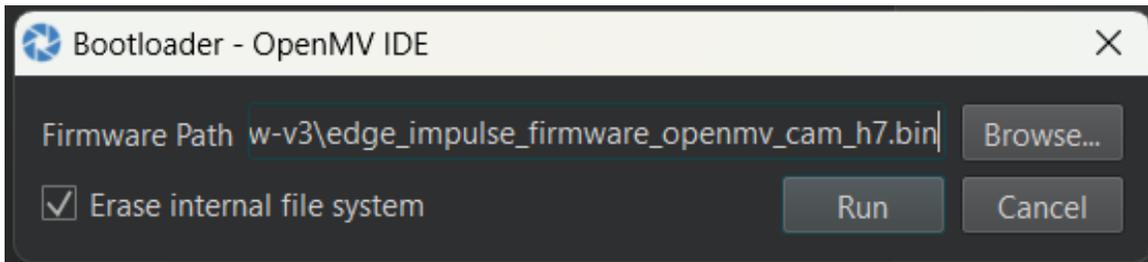


Figura 45: Selección de ruta del *firmware* a utilizar en el IDE de OpenMV.

- Luego de terminar el proceso, deberá de aparecer el siguiente mensaje mostrado en la Figura 46, indicando que el *firmware* se ha actualizado.

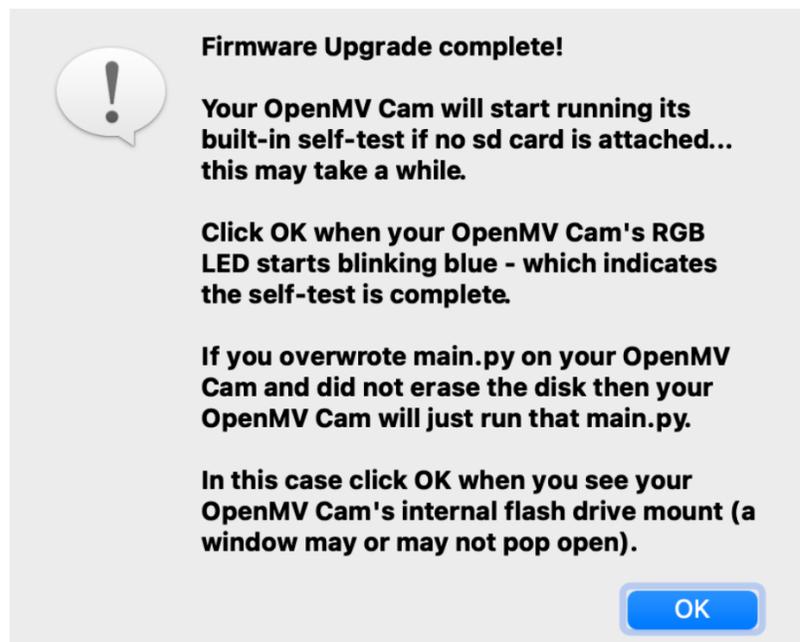


Figura 46: Mensaje de *firmware* actualizado en el IDE de OpenMV.

- Por último, se abre el archivo de código generado anteriormente y se selecciona el botón de **start**, y se puede comenzar las pruebas del modelo entrenado.

8.3.3. Detección de señales de tránsito y estados de semáforo

Se utilizó el archivo de entrenamiento `.tflite` generado anteriormente. Se colocó las configuraciones iniciales para la OpenMV Cam H7, en donde se seleccionó el formato de color RGB565 para poder detectar objetos de color así como un tamaño de resolución de 320x240(QVGA), y se importaron las librerías: `sensor`, `image`, `time`, `os`, `tf`, `math`, `uos` y `gc`, en donde la librería `tf` es la que permite ejecutar los modelos de entrenamiento `.tflite`, como se puede observar en el Código 8.8.

```
1 import sensor, image, time, os, tf, math, uos, gc
2
3 sensor.reset()
4 sensor.set_pixformat(sensor.RGB565)
5 sensor.set_framesize(sensor.QVGA)
6 sensor.set_windowing((240, 240))
7 sensor.skip_frames(time=2000)
```

Código 8.8: Parámetros iniciales de configuración de la cámara.

La función `tf.load` carga el modelo `.tflite` almacenado anteriormente en la memoria de `MicroPython`. Además, calcula que el tamaño del modelo entrenado. Si el tamaño del archivo es mayor que la cantidad de memoria libre menos 64 KB, entonces el modelo se cargará en el búfer de `framebuffer`, de lo contrario mostrara un mensaje de alerta.

```
1 net = tf.load("trained.tflite",
2 load_to_fb=uos.stat('trained.tflite')[6] >
3 (gc.mem_free() - (64*1024)))
```

Código 8.9: Almacenamiento del modelo tflite entrenado.

Luego se carga el archivo `.txt` en el que se encuentran las clases generadas durante el entrenamiento.

```
1 labels = [line.rstrip('\n') for line in open("labels.txt")]
```

Código 8.10: Archivo de clases generadas.

En el bucle se realiza la detección de objetos en tiempo real, la función `net.detect` toma la imagen como entrada y se especifican los umbrales de detección, en donde se coloca la variable `min_confidence` entre 0 y 1, y luego se convierte el límite de los umbrales de 0 a 255, siendo 255 el umbral máximo. Si la detección devuelve el valor 0, significa que la detección es del fondo o no detecta ningún objeto de las clases generadas. Luego se verifica la cantidad de objetos detectados en la lista de detección. Si el valor es igual a 0, el bucle continúa sin realizar nada hasta la siguiente iteración. Si existe una detección de objetos, se imprime en la terminal el nombre de la clase detectada. En el bucle `for`, se obtienen las coordenadas de los objetos detectados con la función `.rect()` desde la esquina superior izquierda. Después, se calcula el centroide del objeto en las coordenadas X y Y y se dibuja un círculo de un color asignado a cada tipo de clase. Esto se puede encontrar en el Código 8.11.

```
1 for i, detection_list in enumerate(net.detect(img,
2 thresholds=[(math.ceil(min_confidence * 255), 255)])):
3     if (i == 0): continue
4     if (len(detection_list) == 0): continue
5
```

```

6      print("***** %s *****" % labels[i])
7      for d in detection_list:
8          [x, y, w, h] = d.rect()
9          center_x = math.floor(x + (w / 2))
10         center_y = math.floor(y + (h / 2))
11         print('x %d\ty %d' % (center_x, center_y))
12         img.draw_circle((center_x, center_y, 12),
13                         color=colors[i], thickness=2)

```

Código 8.11: Detección de las clases entrenadas.

Se utilizó el código y el archivo de entrenamiento .tflite generado. En la Figura 47 se puede observar el IDE de OpenMV en tiempo real, en la terminal se puede observar que el objeto detectado es la señal de ALTO y el centroide en donde se encuentra el objeto en la imagen. En la Figura 48 se puede observar la comparación de una señal de alto positiva detectada 48a) y una señal de alto negativa no detectada 48b), en donde se puede observar que para ser detectada es necesario el color y la forma de un octágono.

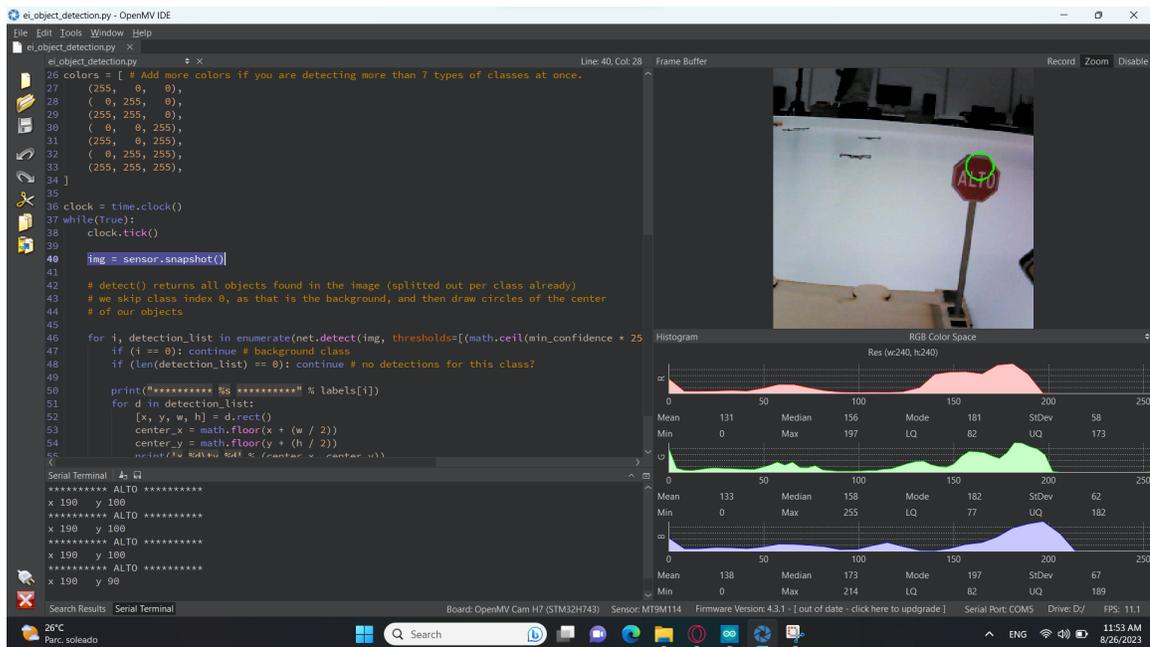
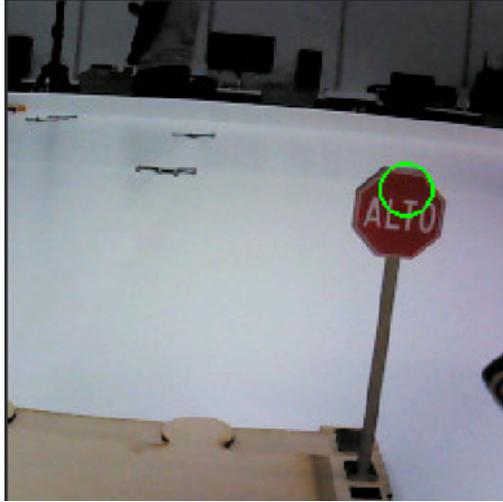


Figura 47: Detección de la señal de alto en el IDE de OpenMV.



(a) Señal de alto positiva



(b) Señal de alto negativa.

Figura 48: Detección de señal de alto.

Se colocó la señal de peatón a escala. En la Figura 49 se puede observar el IDE de OpenMV en tiempo real, donde en la terminal se puede observar que el objeto detectado es la señal de ALTO y el centroide en donde se encuentra el objeto en la imagen. En la Figura 50 se puede observar la comparación de una señal de peatón positiva detectada 50a y una señal de peatón negativa no detectada 50b, en donde se puede observar que para ser detectada es necesario el color y la forma de un rombo.

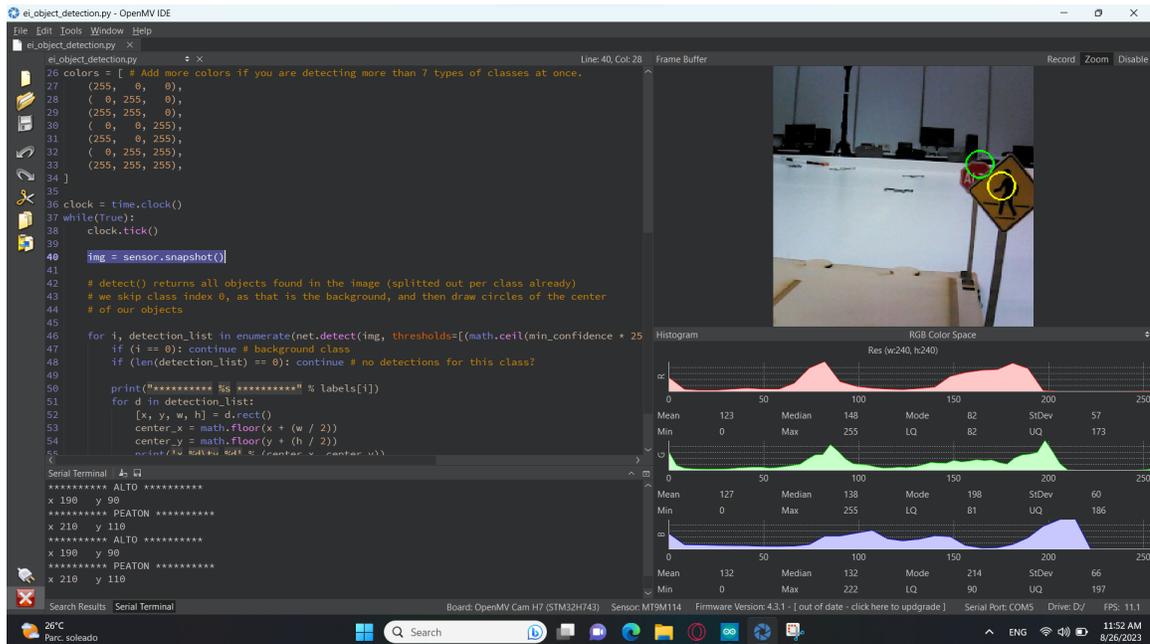
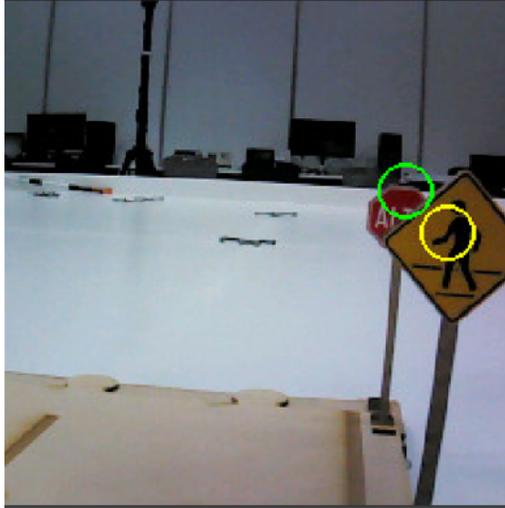


Figura 49: Detección de la señal de peatón en el IDE de OpenMV.



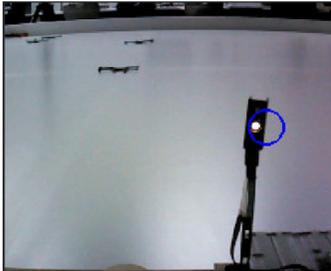
(a) Señal de peatón positiva



(b) Señal de peatón negativa.

Figura 50: Detección de señal de peatón.

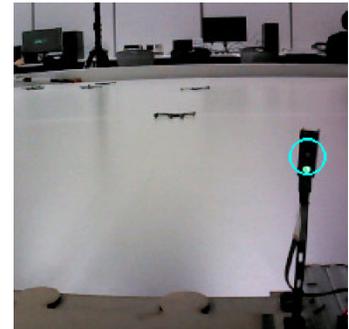
Se colocó el semáforo a escala. En la Figura 51 se puede observar la detección de cada uno de los estados de los semáforos, para lograr la detección de los semáforos se redujo a un 20% la luminosidad de los neopixel, para que la OpenMV Cam H7 lograra detectar los colores. Además, se puede observar que cada una de las clasificaciones es positiva y correcta, lo cual indica que para una clasificación correcta es necesario la forma del semáforo y el color del estado.



(a) Semáforo en amarillo



(b) Semáforo en rojo



(c) Semáforo en verde.

Figura 51: Detección de semáforos.

8.3.4. Protocolo de comunicación

Se utilizó el protocolo de comunicación `UART`. Se importó el módulo `UART` de la librería de PYB. Se utilizó el `UART` número 3, con el TX en el pin 4 y el RX en el pin 5 de la OpenMV Cam H7. Se colocó un *baudrate* de 115200, los datos se envían con un tamaño de 8 bits y en un tiempo máximo de 1000 milisegundos, la configuración se puede observar

en el siguiente fragmento de Código [8.12](#).

```
1  from pyb import UART
2
3  uart = UART(3, 115200, timeout_char=1000) # RX P5, TX P4
4  uart.init(115200, bits=8, parity=None, stop=1, timeout_char=1000)
```

Código 8.12: Inicialización del protocolo de comunicación UART

Se utilizó el formato de [JSON](#): para la transmisión de los datos estructurados. En la detección de la línea amarilla se colocó el valor de θ ρ y en la detección de señales de tránsito se colocó el *label* del objeto detectado. La función `json.dumps` devuelve el objeto como una palabra de [JSON](#): Por último, se envía los datos por [UART](#): esto se puede observar en el siguiente fragmento de Código [8.13](#).

```
1  data = {
2      "theta": line.theta(),
3      "rho": line.rho()
4  }
5
6  json_str_linea = json.dumps(data)
7  uart.write(json_str + '\n')
8
9  json_str_detection = json.dumps(labels[i])
10 uart.write(json_str + '\n')
```

Código 8.13: Envío de datos utilizando JSON

En este capítulo se explica la validación de los algoritmos y modelos de clasificación utilizados en los capítulos anteriores. La validación es esencial para garantizar que el modelo funcione de manera efectiva en el mundo real y pueda tomar decisiones en entornos o características que no fueron considerados durante el entrenamiento.

9.1. Validación de modelos de detección de objetos .tflite

La plataforma de *Edge impulse* proporciona la matriz de confusión de las clases colocadas después de realizar el entrenamiento. En la Figura 52 se puede observar la matriz de confusión del primer modelo entrenado, donde se obtuvo un resultado del 67.3% de exactitud de reconocimiento en el *dataset* de testeo. En el segundo modelo, se redujo el tamaño de los píxeles del tamaño de imagen de reconocimiento de 128X128 píxeles a 96x96 píxeles, lo que resultó en 74.8% de exactitud de reconocimiento en el *dataset* de testeo(Figura 53). Este aumento en el porcentaje se debe a al tamaño del reconocimiento de imagen y el tamaño de píxeles de los objetos en el *dataset* de testeo, ya que al tener un menor tamaño de reconocimiento el modelo puede detectar objetos de menor tamaño en las imágenes.

Se realizó un tercer modelo de entrenamiento, en este modelo se mantuvo la configuración del tamaño de los píxeles de la imagen de reconocimiento y se eliminaron las imágenes del *dataset* en donde el tamaño de los píxeles del objeto eran menor a 96x96 píxeles. En la Figura 54 se puede observar que se obtuvo un resultado del 72.6%.



Figura 52: Matriz de confusión del primer modelo de entrenamiento de 128x128 píxeles.



Figura 53: Matriz de confusión del segundo modelo de entrenamiento de 96x96 píxeles.



Figura 54: Matriz de confusión del tercer modelo de entrenamiento de 96x96 píxeles.

Se realizó una prueba de detección de señales de tráfico a escala utilizando la OpenMV Cam H7 y el primero modelo entrenado. Se programó la cámara para que cuando detecte una señal de tráfico en una imagen, esta sea almacenada en la memoria SD interna. Además, se incluyó en el archivo de código la lógica para etiquetar la imagen detectada con el tipo de señal identificada y para registrar el número de veces que esta señal ha sido detectada hasta el momento. Se realizaron un total de 458 detecciones, con un resultado de 89.3% de predicciones verdaderas positivas. En el Cuadro 5 se puede observar la matriz de confusión experimental, el semáforo verde muestra el mayor porcentaje de predicciones positivas con 96.7% y la señal de alto mostró el menor porcentaje de predicciones positivas con 72.6%, teniendo un porcentaje de predicciones negativas del 21.0% con el semáforo en rojo.

	Alto	Peatón	Semáforo amarillo	Semáforo rojo	Semáforo verde
Alto	72.6%	0.0%	0.0%	0.0%	0.0%
Peatón	0.0%	88.5%	1.7%	0.0%	0.0%
Semáforo amarillo	3.2%	10.3%	93.1%	9.5%	3.3%
Semáforo rojo	21.0%	0.0%	3.4%	89.1%	0.0%
Semáforo verde	3.2%	1.3%	1.7%	1.7%	96.7%

Cuadro 5: Matriz de confusión experimental del primer modelo.

En el Cuadro 6 se puede observar la matriz de confusión del segundo modelo, en donde se realizaron 597 detecciones con un resultado de 93.6% de predicciones positivas, en este

modelo la señal de alto presento el mayor porcentaje de predicciones positivas con un 99.1 % y el semáforo en rojo mostró el menor porcentaje de predicciones positivas con un 82.9%. El segundo modelo mostró un mejor rendimiento de predicciones positivas, esto se puede deber a tener una resolución de tamaño de 96x96 píxeles.

	Alto	Peatón	Semáforo amarillo	Semáforo rojo	Semáforo verde
Alto	99.1 %	0.0 %	0.0 %	1.4 %	0.0 %
Peatón	0.0 %	98.3 %	2.3 %	0.0 %	0.0 %
Semáforo amarillo	0.0 %	1.7 %	95.4 %	15.7 %	5.0 %
Semáforo rojo	0.9 %	0.0 %	1.1 %	82.9 %	0.0 %
Semáforo verde	0.0 %	0.0 %	1.1 %	0.0 %	95.0 %

Cuadro 6: Matriz de confusión experimental del segundo modelo.

Se realizó la medición de la distancia mínima necesaria para la detección de las señales de tránsito y semáforo. En el Cuadro 7 se pueden observar los valores de las distancias mínimas de detección de las clases, la señal de alto y la de peatón son las que muestran distancia de detección de mayor magnitud con 42.0 cm y 40.0 cm respectivamente, mientras que el semáforo amarillo muestra la menor distancia mínima con 16.0 cm.

Señal	Distancia (cm)
Alto	42.0
Peatón	40.0
Semáforo verde	25.0
Semáforo amarillo	16.0
Semáforo rojo	29.0

Cuadro 7: Distancia mínima de detección de señales de tránsito.

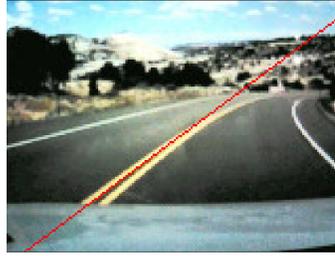
9.2. Validación de la detección de carriles

Para la validación de la detección de los carriles con amarilla en los se utilizó escenas en movimiento en carreteras con distintos ambientes y momentos del día, entre ellos se encuentra la carretera hacia Utah [30], la ruta estatal 123 en el parque nacional Pinchot [31], la carretera hacia el parque nacional Yosemite [32] y la ciudad de Seúl de noche [33]. Se utilizó el IDE de OpenMV para visualizar el algoritmo utilizado.

La carretera hacia Utah es de doble carril y sentidos opuestos con un carril de color amarillo en medio de ellos, el ecosistema alrededor de de la carretera es desértico, terrenos rocosos y con vistas a los cañones y formaciones rocosas, se puede observar que es de día con el cielo despejado e iluminación del sol. En la Figura 55 se puede observar figuras de las detecciones a lo largo de la carretera. En la Figura 58a se puede observar la detección de un carril recto, en la Figura 58b se puede observar la línea de color rojo alineándose en el centro de la curva con dirección hacia la izquierda, en la Figura 58c se observa la alineación en el centro de la curva con dirección hacia la derecha.



(a) Carril recto.



(b) Carril con curva hacia la izquierda.



(c) Carril con curva hacia la derecha.

Figura 55: Detección de carriles en carretera hacia Utah.

La carretera hacia el parque nacional Yosemite es de doble carril y con sentidos opuestos con un carril de color amarillo en el medio de ellos, el ecosistema alrededor es boscoso, con presencia de arboles, arbustos y grama en su alrededor y de día como se puede observar en las detecciones de carril de la Figura 56. En las Figuras 56a y 56b se puede observar la detección y la alineación del carril recto en este ecosistema, además, se puede observar en las Figuras 56b y 56c que con la presencia de vehículos y ciclistas continua realizando la detección de carril y la alineación.



(a) Carril recto.



(b) Carril con ciclista



(c) Carretera con vehículos

Figura 56: Detección de carriles en carretera hacia el parque nacional Yosemite

La carretera hacia la ruta estatal 123 es de dos carriles y vias opuestas, en el parque nacional Pinchot. Esta carretera muestra vistas de montañas, vegetación en la época de primavera. Tiene un clima despejado y de día como se puede observar en las detecciones de las Figuras 57. En las Figuras 57a y 57b se puede observar la detección de carriles rectos, en la Figura 57c se puede observar la detección y la alineación del carril con curva con dirección hacia la derecha.

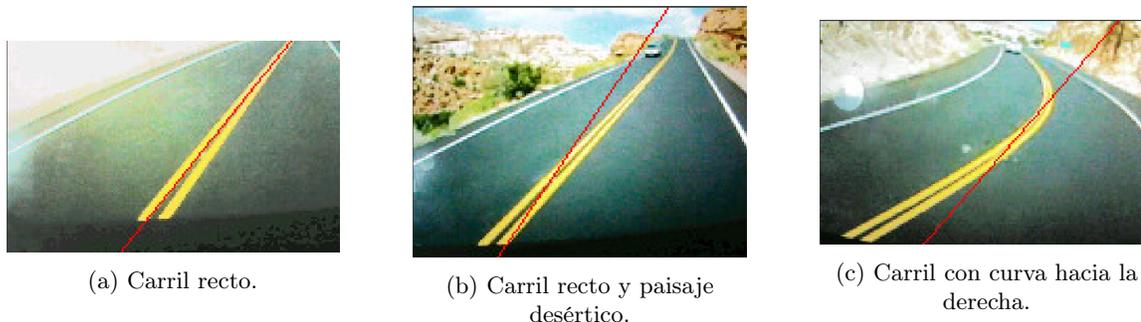


Figura 57: Detección de carriles en carretera hacia la ruta estatal 123 en el parque nacional Pinchot.

La carretera en la ciudad de Seúl es de dos a tres carriles en algunos segmentos en cada una de las vías, es de noche por lo que muestra iluminación, semáforos, señales de tránsito, vehículos y motocicletas. En las detecciones de las Figuras 58 se puede observar la alineación de los carriles amarillos con iluminación artificial proporcionada por el vehículos y postes de luz. Se puede validar la detección y alineación de carriles amarillos de noche con iluminación artificial.

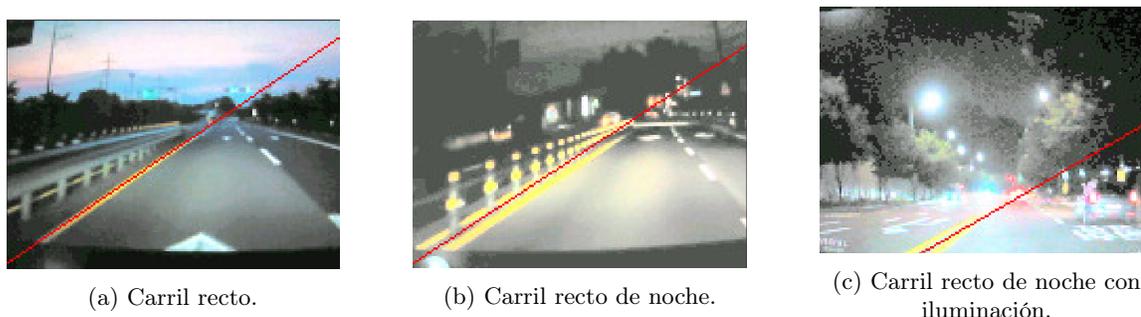


Figura 58: Detección de carriles en carretera de noche en la ciudad de Seúl.

Para la detección de los carriles amarillos en las carreteras a escala se pintó de color gris para crear un contraste de color entre la superficie y el carril. Luego de validar la detección en la carretera a escala pintada se utilizó el algoritmo de detección de carril y se implementó con el controlador de seguidor de línea del trabajo de graduación de Fernando Arribas 34, en donde se colocó en la infraestructura escala y el algoritmo de detección en el Polulu 3pi, en la Figura 59 se puede observar los fotogramas del movimiento del vehículo a escala y el seguimiento y corrección del carril amarillo en la recta y curva de la carretera.



Figura 59: Fotogramas del algoritmo detección de carril amarillo implementado.

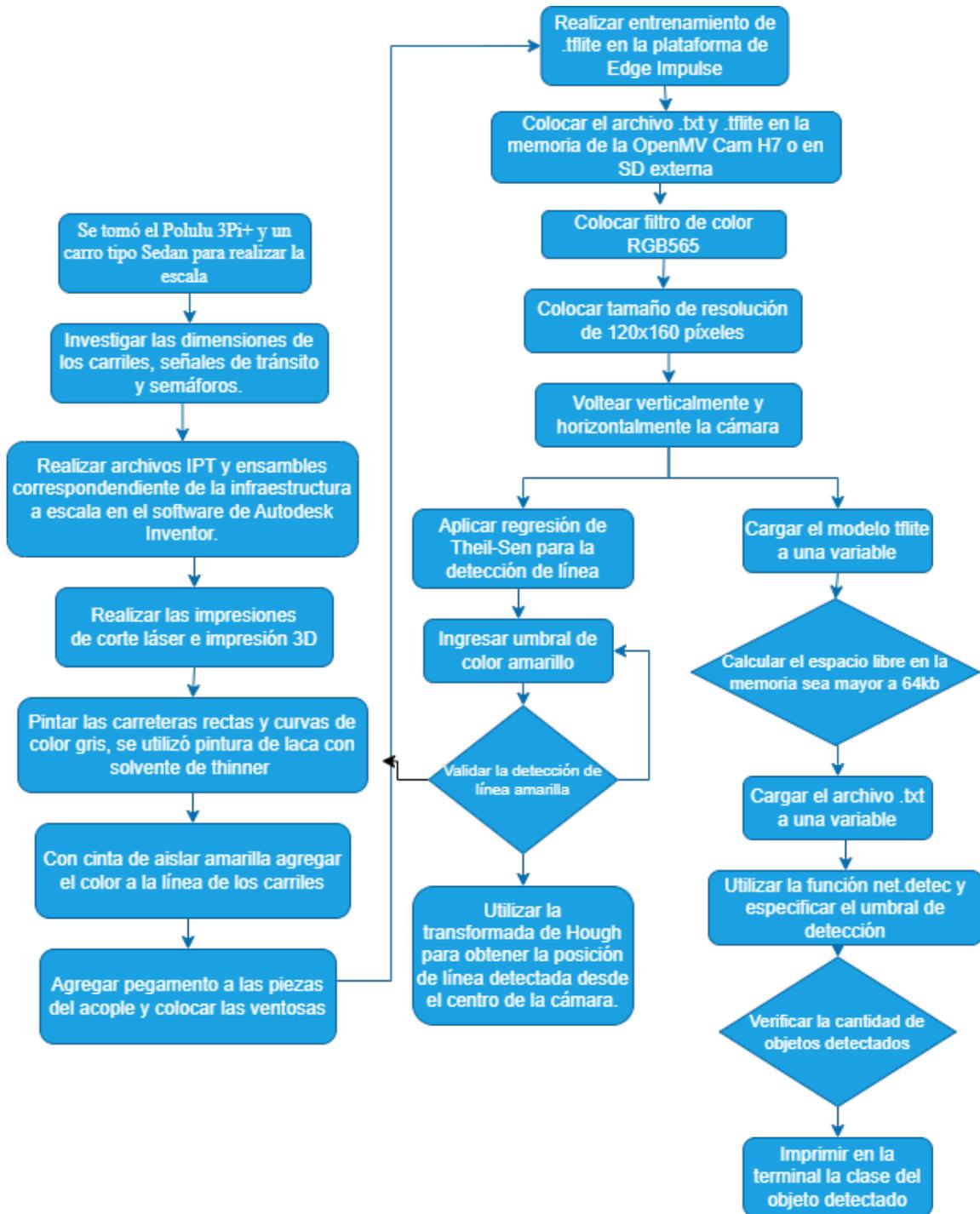


Figura 60: Diagrama de solución planteada.

- Se diseñó y fabricaron 132 carreteras de dos carriles, señales de tránsito y semáforos, con una escala de reducción de 1:18.75 obtenida mediante la relación del vehículo modelo sedán de un Honda y el Polulu 3pi+.
- Se implementó un algoritmo de detección de líneas capaz de detectar líneas de color amarillo y alinearlas verticalmente, utilizando la transformada Hough.
- La plataforma de Edge Impulse permitió entrenar y optimizar modelos de *machine learning* con un rendimiento de 25 FPS y ocupando un 25% de la memoria de la OpenMV Cam H7.
- Se implementó de un modelo TensorFlow Lite de detección de objetos de 5 clases (señal de alto, señal de peatón y estados del semáforo), con porcentaje de validación máxima de 93.6% y mínima de 89.3%.

CAPÍTULO 11

Recomendaciones

- Se recomienda utilizar el modelo OpenCam H7 Pro, para poder utilizar modelos con mayor capacidad de procesamiento en la memoria RAM, lo que permitiría realizar aplicaciones de mayor tamaño en términos de cálculos y almacenamiento.
- Se recomienda que los objetos a detectar en la imágenes utilizadas en el *dataset* tengan un tamaño mínimo de 96x96 píxeles, para que puedan ser detectadas en el testeo del modelo de entrenamiento.
- Se recomienda tener un contraste de color entre la base de la carretera y la línea de los carriles, para realizar una correcta detección de la línea de los carriles.

-
- [1] “*Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*,” Society of Automotive Engineers, Pensilvania, USA, Standard, dic. de 2018.
 - [2] KPMG, “*2020 Autonomous Vehicles Readiness Index*,” KPMG, inf. téc., 2020.
 - [3] S. Lopez C. Espinosa y K. Villareal, “Diseño e implementación de un prototipo de vehículo autónomo a escala con capacidades de visión artificial,” Tesis de licenciatura, Universidad San Francisco de Quito, 2021.
 - [4] J. Á. Balbuena, “Sistema de visión por computadora para la detección de las líneas del carril y cruce, aplicado a una plataforma autónoma móvil a escala: AutoModelCar,” Tesis de licenciatura, Instituto Tecnológico Superior de Atlixco, 2020.
 - [5] D. Quiroga y B. Rubio, “Robot Móvil Autónomo para la Siembra de Semillas en el Campo,” Tesis de licenciatura, Universidad Politécnica Salesiana de Ecuador, 2023.
 - [6] S. Waslander y J. Kelly. “*Visual Perception for Self-Driving Cars*,” Coursera. (2023), dirección: <https://www.coursera.org/learn/visual-perception-self-driving-cars?specialization=self-driving-cars>.
 - [7] R. Szeliski, *Computer Vision: Algorithms and Applications*, 2st. Springer, 2021.
 - [8] J. Martínez, “Reconocimiento de imágenes mediante redes neuronales convolucionales,” Tesis de licenciatura, Universidad Politécnica de Madrid, 2020.
 - [9] Á. Artola, “Clasificación de imágenes usando redes neuronales convolucionales en Python,” Tesis de licenciatura, Universidad de Sevilla, 2019.
 - [10] C. Szegedy, W. Liu, Y. Jia et al., “*Going Deeper with Convolutions*,” en *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.
 - [11] O. Russakovsky, J. Deng, H. Su et al., “*ImageNet Large Scale Visual Recognition Challenge*,” *International Journal of Computer Vision*, 2015.
 - [12] A. Krizhevsky, “*Learning multiple layers of features from tiny images*,” *Canadian Institute for Advanced Research*, 2019.

- [13] L. Fei-Fei, “*ImageNet: A Large-Scale Hierarchical Image Database*,” en *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009.
- [14] Á. Rayo, “Estudio del módulo “OpenMV Cam H7” para aplicaciones de visión artificial,” Tesis de licenciatura, Universidad de Alcalá Escuela Politécnica Superior, 2021.
- [15] OpenMV. “*class HaarCascade – Feature Descriptor*.” (2023), dirección: <https://docs.openmv.io/library/omv.image.html>.
- [16] TensorFlow, “TensorFlow Lite,” TensorFlow, inf. téc., 2023. dirección: <https://www.tensorflow.org/lite/guide>.
- [17] OpenMV, “TensorFlow,” OpenMV, inf. téc., 19 de julio de 2023. dirección: <https://docs.openmv.io/library/omv.tf.html>.
- [18] E. Impulse, “*Getting Started*,” Edge Impulse, inf. téc., 2023. dirección: <https://docs.edgeimpulse.com/docs/>.
- [19] G. Ortiz Zamora, “Procesamiento morfológico de imágenes en color: aplicación a la reconstrucción geodésica,” Tesis de licenciatura, Universidad de Alicante, 2002.
- [20] R. Ladero, “Detección de señales y líneas de carril para la conducción autónoma con vehículo a escala,” Tesis de licenciatura, Universidad Politécnica de Madrid, 2019.
- [21] scikit-learn. “Regresión de Theil-Sen.” (2023), dirección: https://qu4nt.github.io/sklearn-doc-es/auto_examples/linear_model/plot_theilsen.html.
- [22] OpenMV, “OpenMV Cam H7,” OpenMV, inf. téc., 2023.
- [23] P. Corporation. “*Micro Metal Gearmotor 6V 100RPM*.” (2023), dirección: <https://www.pololu.com/product/4974>.
- [24] Adafruit. “*The Magic of NeoPixels*.” (2023), dirección: <https://learn.adafruit.com/adafruit-neopixel-uberguide/the-magic-of-neopixels>.
- [25] M. R. Durán Ortiz, *Manual Centroamericano de Dispositivos Uniformes para el Control del Tránsito: Catálogo de Señales*, S. de Integración Económica Centroamericana, ed. Secretaría de Integración Económica Centroamericana, 2014.
- [26] “Conector JST SM de 3 pines.” (2023), dirección: <https://tienda.tettsa.gt/wp-content/uploads/2023/07/Conector-JST-SM-3-pines.jpg>.
- [27] Adafruit. “Adafruit NeoPixel.” (2023), dirección: https://github.com/adafruit/Adafruit_NeoPixel.
- [28] M. Reyes. “*Stop Sign Detection using OpenCV*.” (9 de julio de 2018), dirección: https://github.com/maurehur/Stop-Sign-detection_OpenCV.
- [29] A. Cordolino. “*Vehicle detection haarcascades*.” (2015), dirección: https://github.com/andrewssobral/vehicle_detection_haarcascades/blob/master/cars.xml.
- [30] 4. R. Channel. “*4K Scenic Byway 12 | All American Road in Utah, USA - 5 Hour of Road Drive with Relaxing Music*.” (2018), dirección: <https://www.youtube.com/watch?v=Z0Z0qbK86t0&t=9409s>.
- [31] AdventureEveryDay. “*Rocky Mountain National Park 4K | Estes Park to Grand Lake | Trail Ridge Road Complete Scenic Drive*.” (2022), dirección: <https://www.youtube.com/watch?v=CiTg-iYVBYo&t=290s>.

- [32] AdventureEveryDay. “*Tioga Pass Scenic Drive Through Yosemite National Park - Sierra Nevada Mountains 4K.*” (2023), dirección: <https://www.youtube.com/watch?v=qqrdoJ9fj44&t=448s>.
- [33] caminante de seúl | Seoul Walker. “*Seoul Night Drive| Downtown Vibes | Chill Lofi Hip-hop Beats | POV 4K HDR.*” (2022), dirección: <https://www.youtube.com/watch?v=SRpMapyw6Aw&t=557s>.
- [34] F. Arribas Valdez, “Pruebas a escala de algoritmos básicos de visión de computadora y control para vehículos autónomos a escala,” Tesis de licenciatura, Universidad del Valle de Guatemala, 2023.

13.1. Diseños de infraestructura a escala

Este enlace dirige a la carpeta que contiene los diseños de la infraestructura a escala, como IPT, ensamblados, STL y archivos para corte láser https://drive.google.com/drive/folders/1kWnTPn9PGVVWBtXvu-P8lqo87jqsFzq_?usp=sharing.

13.2. Detección de carriles en escenas de carreteras

Este enlace dirige a la carpeta en donde se encuentran las pruebas en las distintas escenas de detección de carril en las carreteras <https://drive.google.com/drive/folders/1Mts10Rx9e-rqkK1KM9b5wQYpt93a09CZ?usp=sharing>.

13.3. Dataset de señales de tránsito

Este enlace dirige a la carpeta en donde se encuentran las imágenes utilizadas para el entrenamiento de clasificación de señales de tránsito y semáforos https://drive.google.com/drive/folders/1UtXo83giX0GIvwi5ZuChjKDFqW65bp?usp=drive_link.

13.4. Repositorio de códigos de los algoritmos de visión por computados

Este enlace dirige al repositorio de los algoritmos de detección de línea, algoritmos de detección y clasificación de señales de tránsito y semáforos y el control de los semáforos https://github.com/GAFong/Algoritmos_vision_computador.git.

Autodesk Inventor: Es un software de diseño asistido por computadora (CAD) desarrollado por Autodesk. Está diseñado principalmente para la creación de modelos 3D, simulación, renderización y documentación en el campo de la ingeniería mecánica, diseño de productos y fabricación.. [20](#), [30](#), [31](#)

Google street view: Es una herramienta de Google Maps que permite a los usuarios explorar el mundo desde la comodidad de su hogar. Ofrece imágenes panorámicas de 360 grados de lugares de todo el mundo, desde calles y carreteras hasta parques y atracciones turísticas.. [43](#)

JSON: Es una herramienta de Google Maps que permite a los usuarios explorar el mundo desde la comodidad de su hogar. Ofrece imágenes panorámicas de 360 grados de lugares de todo el mundo, desde calles y carreteras hasta parques y atracciones turísticas.. [54](#)

MicroPython: Es una implementación del lenguaje de programación Python 3, escrita en C, optimizada para poder ejecutarse en un microcontrolador.. [50](#)

PWM: (Modulación por Ancho de Pulso, por sus siglas en inglés *Pulse Width Modulation*) es una técnica que se utiliza para controlar la cantidad de energía que se envía a una carga, modificando el ciclo de trabajo de una señal periódica. El ciclo de trabajo es la relación entre el tiempo que está activa la señal y el tiempo total de la señal. . [36](#)

Transfer learning: es un enfoque en el campo del aprendizaje automático y la inteligencia artificial que implica transferir el conocimiento adquirido al entrenar un modelo de una tarea específica a otra tarea relacionada. En lugar de entrenar un modelo desde cero para cada tarea individual, aprovecha el aprendizaje obtenido de una tarea inicial y lo aplica para mejorar el rendimiento en una tarea diferente pero relacionada.. [7](#), [45](#)

UART: es un protocolo de comunicación serial asíncrono que se utiliza para transmitir datos entre dos dispositivos. . [53](#), [54](#)