2023-04

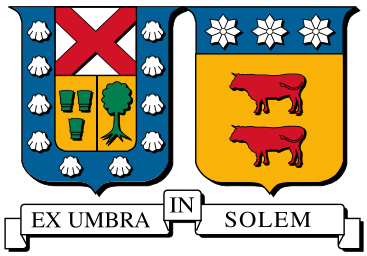
Implementación y evaluación de algoritmos de detección de carril orientados a entornos de conducción asistida de vehículos

Espejo Miranda, Ignacio Andrés

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARIA

DEPARTAMENTO DE ELECTRÓNICA

VALPARAISO - CHILE



" IMPLEMENTACIÓN Y EVALUACIÓN DE ALGORITMOS DE DETECCIÓN DE CARRIL ORIENTADOS A ENTORNOS DE CONDUCCIÓN ASISTIDA DE VEHÍCULOS "

IGNACIO ANDRÉS ESPEJO MIRANDA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRÓNICO, MENCIÓN COMPUTADORES

PROFESOR GUÍA: DR. GONZALO CARVAJAL

PROFESOR CORREFERENTE: DR. FRANCISCO VARGAS

Abril 2023

DEPARTAMENTO DE ELECTRONICA
UNIVERSIDAD TECNICA FEDERICO SANTA MARIA

DEPARTAMENTO DE ELECTRONICA

Implementación y evaluación de algoritmos de detección de carril orientados a entornos de conducción asistida de vehículos

Ignacio Andrés Espejo Miranda

Memoria para optar al título de Ingeniero Civil Electrónico

Universidad Técnica Federico Santa María

Profesor Guía: Dr. Gonzalo Carvajal

Abril 2023

Resumen

El desarrollo de vehículos autónomos ha capturado el interés de grandes empresas como Google y Tesla debido a los desafíos tecnológicos que presentan. Los vehículos autónomos requieren sensores y algoritmos capaces de percibir el ambiente en el cual se encuentran inmersos, y uno de los primeros desafíos es detectar las líneas que delimitan los carriles dentro de la carretera. El diseño de un detector de carriles es un problema desafiante, y existen alternativas de diseño basadas en utilizar uno o más sensores LiDAR o una cámara como sensor junto con algoritmos de visión por computadora. Los desafíos asociados con el desarrollo de sistemas de conducción autónoma y asistida incluyen altos costos económicos para probar el rendimiento de los algoritmos en vehículos reales. Una de las alternativas implica la evaluación del rendimiento de los algoritmos en modelos urbanos a escala junto con robots móviles de bajo costo, como es la plataforma a escala RUPU desarrollada por profesores y estudiantes del Departamento de Electrónica de la Universidad Técnica Federico Santa María.

Esta memoria se enfoca en el estudio, implementación y evaluación

de dos algoritmos de detección de carril (tradicional y basado en deep learning) para su utilización en una futura iteración de la plataforma RUPU. Se evaluó la capacidad de los algoritmos para detectar carriles y su latencia en diferentes escenarios. La capacidad de los algoritmos para detectar carriles se evalúa cualitativamente mediante visualizaciones de las salidas de los detectores en diferentes escenarios, debido a que recientemente las métricas cuantitativas comúnmente utilizadas para medir el rendimiento de los detectores de carriles han sido cuestionadas como no aptas para tareas orientadas a la conducción de vehículos.

Los datos obtenidos muestran que actualmente el detector tradicional es preferible y se recomienda desarrollarlo en conjunto con la plataforma RUPU, utilizando una zona de interés para disminuir falsos positivos, líneas delimitadoras de carril de diferentes colores en la superficie de RUPU y robots móviles y obstáculos de colores diferentes a los utilizados en las líneas. El detector puede obtener valores de latencia en el orden de magnitud de 10 [ms] en una tarjeta Jetson TX2 sin utilizar unidades de procesamiento paralelo.

Abstract

The development of autonomous vehicles has captured the interest of large companies such as Google and Tesla due to the technological challenges they present. Autonomous vehicles require sensors and algorithms capable of perceiving the environment in which they are immersed, and one of the first challenges is to detect the lines that delimit the lanes on the road. The design of a lane detector is a challenging problem, and there are design alternatives based on using one or more LiDAR sensors or a camera as a sensor together with computer vision algorithms. The challenges associated with the development of autonomous and assisted driving systems include high economic costs to test the performance of the algorithms in real vehicles. One of the alternatives involves evaluating the performance of the algorithms in urban scale models together with low-cost mobile robots, such as the RUPU scale platform developed by professors and students of the Electronics Department of the Federico Santa María Technical University.

The memory focuses on the study, implementation and evaluation of two lane detectors (traditional and based on deep learning) for use in a future iteration of the RUPU platform. The ability of the algorithms to detect lanes and their latency in different scenarios was evaluated. The ability of the algorithms to detect lanes is evaluated qualitatively by visualizing the detector outputs in different scenarios, because recently the quantitative metrics commonly used to measure the performance of lane detectors have been questioned as not suitable for task oriented to driving vehicles.

The data obtained show that currently the traditional detector is

preferable and it is recommended to develop it in conjunction with the RUPU platform, using an area of interest to reduce false positives, lane delimiting lines of different colors on the RUPU surface and mobile robots and road obstacles. colors different from those used in the lines. The detector can achieve latency values in the order of magnitude of 10 [ms] on a Jetson TX2 card without using parallel processing units.

Índice de contenidos

Índice de figuras

1. Introducción

El presente documento describe el trabajo realizado para la implementación y evaluación de algoritmos de detección de carriles basados en técnicas tradicionales de procesamiento de imágenes y *Deep Learning*.

Este capítulo presenta el contexto y la motivación para el desarrollo de este trabajo, especifica el problema a resolver, los objetivos, alcances y contribuciones de esta memoria de título, y finalmente presenta la organización del resto del informe.

1.1. Motivación y contexto

Durante los últimos años, el desarrollo de vehículos autónomos se ha vuelto un tópico de investigación y desarrollo muy popular en contextos académicos e industriales [22], presentando importantes desafíos tecnológicos que han capturado el interés de grandes empresas como Google[23] y Tesla[24], entre otras. Para el caso de un automóvil autónomo que se desplaza por una carretera, este requiere sensores[25] y algoritmos capaces de percibir el ambiente en el cual el vehículo se encuentra inmerso, extrayendo información tal como la curvatura del carril, presencia de señales de tránsito, obstáculos, peatones, otros vehículos, entre otros, con el fin de que esta información sea utilizada por los sistemas de control del vehículo, encargados de regular su velocidad y dirección.

Para que un vehículo autónomo sea capaz de desplazarse por una carretera, es fundamental que este sea capaz de avanzar manteniéndose dentro del carril indicado(*lane keeping*), para lo cual una de las primeras tareas que debe satisfacer es percibir las líneas que delimitan los carriles dentro de la carretera. Los algoritmos encargados de percibir estas líneas son llamados detectores de carriles. El diseñar un detector de carriles representa un problema desafiante, con múltiples alternativas de sensores disponibles para su desarrollo, a los cuales se debe adecuar el detector, además de que debe cumplir con requerimientos de tiempo de reacción limite asociados a la seguridad de los ocupantes del vehículo para entregar oportunamente la información a los sistemas de control.

Por ejemplo, el tiempo para desplegar un sistema de *airbag* luego de un choque lateral está en el rango de 10 a 20[ms][26], y por ende requieren de cómputo especializado con sistemas embebidos específicamente diseñados para cumplir su función en plazos específicos.

Con respecto a la implementación de sistemas de detección de carril, existen alternativas de diseño basadas en utilizar uno o más sensores LiDAR[27–29] para detectar la posición de las marcas delimitadoras de carriles. Por ejemplo, al utilizar las capacidades de detección activa de rango de luz se puede detectar las marcas delimitadoras blancas altamente reflectivas en una carretera[30]. Otra opción de diseño de detector de carril es utilizar una cámara como sensor junto con algoritmos de visión por computador. Esta última opción ha sido utilizada frecuentemente ya que al menos una cámara se encuentra disponible en diversos vehículos, siendo consideradas como fundamentales por muchos fabricadores de vehículos[25] y se han desarrollado plataformas de aprendizaje en base a esta alternativa debido a su simplicidad, su capacidad de percibir colores y texturas, y su versatilidad, ya que una imagen puede ser procesada posteriormente por múltiples algoritmos con diferentes fines.

En la literatura existen múltiples detectores de carriles basados en cámaras tradicionales que pueden ser clasificados como detectores tradicionales[12, 31, 32] o basados en *deep learning*[13, 14, 17, 33]. Los detectores tradicionales utilizan operaciones de visión por computador clásicas como la Transformada de Hough o el Detector de bordes Canny, que son ajustados manualmente para extraer características de la imagen, como los bordes de las líneas delimitadoras de carril. Los algoritmos tradicionales tienen un costo computacional relativamente bajo y pueden ser utilizados efectivamente sin el uso de arquitecturas paralelas, pero no son los suficientemente robustos como para lidiar con la diversidad de carriles en los diferentes escenarios de un ambiente real[15] debido a sus operaciones ajustadas manualmente y por ende en el último tiempo la mayoría de los estudios se han centrado en detectores basados en *deep learning*. Los detectores basados en *deep learning* utilizan capas de redes neuronales artificiales, como redes

convolucionales por ejemplo, para resolver el problema de localización de los bordes del carril. Los detectores basados en *deep learning* requieren el almacenamiento de millones de parámetros y el cálculo de múltiples millones de operaciones aritméticas, por lo que se benefician enormemente del uso de arquitecturas paralelas, capaces de realizar múltiples operaciones en paralelo como una unidad de procesamiento gráfico (GPU).

Si bien la cantidad de artículos sobre detectores de carril es abundante, muchos de estos se centran en obtener la mayor precisión posible en *datasets* específicos [14, 33–35], sin tener en consideración el alto costo computacional de sus algoritmos, lo cual no los hace aptos para su uso generalizado dentro de una carretera real. No todos los artículos facilitan el código de sus algoritmos[36] para replicar sus resultados. Aquellos que lo facilitan frecuentemente poseen un proceso de instalación complicado, con dependencias que generan conflictos de versiones, junto con escasas instrucciones que los hacen difíciles implementar y adaptar a distintos ambientes de desarrollo. Estos obstáculos dificultan replicar los experimentos y aplicar los algoritmos a nuevos casos de estudio.

En el contexto de vehículos autónomos o de conducción asistida, los detectores de carriles están integrados dentro de un sistema (ADAS) y son utilizados para entregar información acerca del ambiente a los algoritmos de control, encargados de controlar los actuadores del vehículo. Un desafío asociado al desarrollo de estos sistemas es el alto costo económico de probar el rendimiento de los algoritmos en un vehículo como un automóvil, requiriendo un capital capaz de pagar los costos de compra del vehículo, las modificaciones en su configuración de manejo, actuadores, múltiples sensores adicionales, computadores con la suficiente capacidad de cómputo y el alto costo de las reparaciones en caso de que el vehículo se dañe. Por esta razón se suele utilizar como alternativa la evaluación de rendimiento de los algoritmos en modelos urbanos a escala junto con robots móviles de bajo costo[16].

El desarrollo de esta memoria se enfoca a un futuro uso de la plataforma a escala RUPU[1], desarrollada por profesores y estudiantes del Departamento de Electrónica de la Universidad Técnica Federico Santa María para el despliegue y la prueba experi-

mental de estrategias para tratar los problemas de control lateral y longitudinal en un pelotón de vehículos que siguen una ruta. La plataforma actualmente está enfocada al seguimiento de línea, pero en futuras iteraciones estas serán reemplazadas por carriles. La plataforma actualmente consta de dos elementos: (i) una superficie con una línea a seguir y (ii) un conjunto de agentes móviles electromecánicos equipados con el hardware de detección y computación necesario para seguir la línea en el camino. A continuación se describen la superficie y los agentes móviles, respectivamente:

- La superficie con el camino a seguir corresponde a un juego de tapetes cuadrados ensamblados de espuma EVA negra con líneas blancas pintadas sobre la superficie demarcando el camino. Las líneas de cada tapete pueden ser rectas, curvas o intersecadas y pintadas, de modo que diferentes entrelazados de tapetes puedan producir diferentes patrones de líneas en la superficie. Esta versatilidad nos permite construir fácilmente varias trayectorias de caminos con el mismo grupo de tapetes. La versión actual considera tapetes cuadrados individuales con dimensiones $29 \times 29 \times 0{,}8$ cm.
- Cada agente se implementa como un robot móvil de accionamiento diferencial (DDMR), que puede moverse de forma autónoma sobre la superficie descrita anteriormente. La figura 1.1 muestra una descripción general de los componentes de un agente. El agente se mueve a través de dos ruedas activas, que se controlan de forma independiente, lo que permite que los agentes avancen, retrocedan, giren, giren o mantengan una trayectoria curva. El núcleo de procesamiento central para cada agente es una placa de desarrollo integrada ESP32 de bajo costo, que está equipada con un microprocesador LX6 de 32 bits de doble núcleo Xtensa.

Los robots móviles de RUPU utilizan un procesador de bajo costo, el cual tiene una capacidad de cómputo limitada. Además, los modelos urbanos a escala diseñados para estos robots representan un entorno controlado, a diferencia de los entornos reales representados en los *datasets* utilizados en la literatura, cambiando las condiciones de operación de los algoritmos, por lo que no se puede garantizar que lo reportado en la

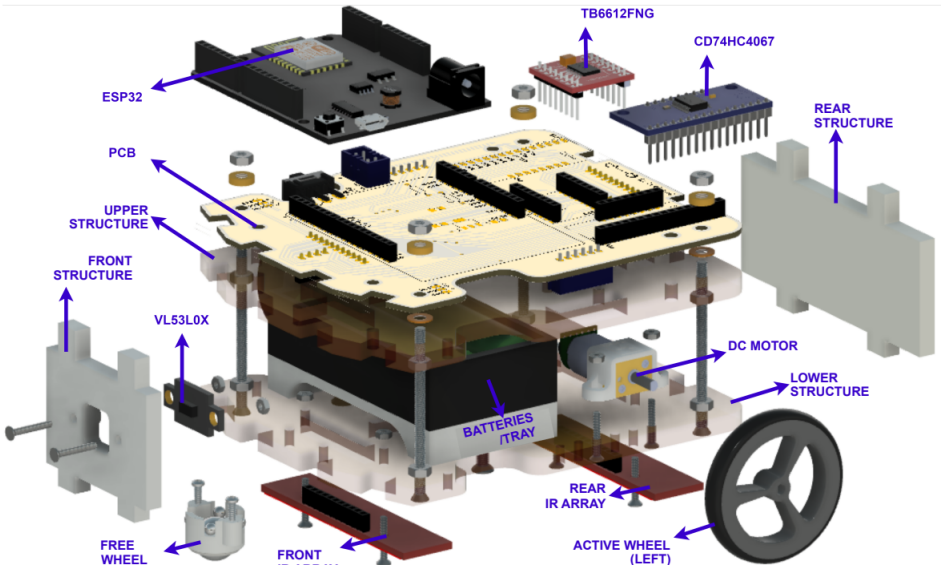


Figura 1.1: Agente de RUPU. Fuente: [1]

literatura funcione.

El trabajo desarrollado en esta memoria de título surge a partir del interés por parte de los profesores Dr. Gonzalo Carvajal y Dr. Francisco Vargas del Departamento de Electrónica de la Universidad Técnica Federico Santa María, de extender la plataforma RUPU para que sus robots móviles sean capaces de seguir carriles en lugar de líneas . En este contexto, es necesario, en primer lugar, que los robots sean capaces de detectar los carriles presentes en la plataforma de RUPU y por ende se exploran las técnicas descritas en la literatura actual que podrían cumplir esta función en modelos a escala.

Esta memoria entonces apunta a la implementación y evaluación de desempeño de algoritmos de detección de carril orientados a su utilización en una plataforma a escala que opera en entorno controlado. Esto como primer paso para determinar los requerimientos y la factibilidad de integrar sistemas de detección de carril en la plataforma RUPU para el estudio de pelotones de vehículos utilizando modelos a escala.

1.2. Planteamiento del problema

El problema por tratar en este trabajo de título es el estudio, implementación, y evaluación de algoritmos de detección de carriles para ser utilizados en la plataforma RUPU. Estos algoritmos reciben una imagen como entrada y entregan como salida las coordenadas de las líneas delimitadoras de carriles presentes en la imagen. Y deben cumplir con los siguientes requisitos para poder funcionar adecuadamente en la plataforma RUPU: bajo costo computacional, baja latencia o tiempo de inferencia, y ser capaz de reconocer líneas delimitadoras de carriles marcadas en la superficie del modelo a escala. Basándose en los análisis realizados en las etapas previas al desarrollo de esta memoria, se selecciona implementar un detector de carril tradicional y uno basado en *deep learning* que han sido diseñados para ser utilizados en modelos a escala generales. Al implementarlos se busca obtener latencias al menos comparables a otros algoritmos similares en la literatura, y son evaluados experimentalmente en base a su latencia junto con visualizaciones en diferentes entornos de conducción (controlado o real) para otorgar una comparación de su rendimiento. Estos algoritmos serán documentados, describiendo el proceso de instalación de dependencias para su funcionamiento, permitiendo a un usuario posterior elegir cual de estos se adecua mejor a sus necesidades y pueda implementarlo sin conflictos de versiones.

El trabajo en torno a la implementación y evaluación de los algoritmos propuestos considera las siguientes etapas:

- **Implementación de algoritmos en computador de escritorio:**

  Se programa un algoritmo de detección de carriles tradicional basado en operaciones ajustadas manualmente para extraer características de la imagen, tales como la transformada de Hough o el detector de bordes Canny. También se programa un detector basado en *deep learning* utilizando redes neuronales completamente conectadas y convolucionales, entre otras. Ambos algoritmos son implementados en un computador de escritorio. Esto con el fin de tener dos algoritmos con diferentes

diseños que son evaluados preliminarmente en un sistema embebido.

- **Evaluación de rendimiento en computador de escritorio:** Se mide experimentalmente la latencia de ambos algoritmos en un computador de escritorio con vídeos grabados en diferentes entornos de conducción, considerando ambientes realistas, como un vídeo de conducción por carretera, o ambientes controlados como una pista diseñada para robots móviles seguidores de línea. Se mide la latencia de procesamiento de un cuadro, o tiempo de inferencia, de ambos algoritmos en los diferentes vídeos y se crea un vídeo de la visualización de los carriles detectados. Los resultados obtenidos en este paso permiten decidir cuál de los algoritmos es factible de implementar en un sistema embebido.
- **Implementación y evaluación de rendimiento en un procesador embebido:** En caso de considerarse adecuado, se implementa el algoritmo en un sistema embebido, realizando cambios en dependencias o transformaciones de modelos si es necesario, y se miden nuevamente las métricas definidas anteriormente. Los resultados reportados en este paso entregan información que nos permite obtener conclusiones acerca de si es factible utilizar este algoritmo en la plataforma RUPU.
- **Creación de repositorio y código de demostración:** Se crean repositorios que contienen el código de ambos algoritmos implementados, junto con instrucciones para su uso e instalación. Además, se crea código que permite procesar un vídeo y mostrar el resultado de salida de ambos algoritmos al mismo tiempo. Este programa apunta a servir como demostrador para ilustrar las capacidades de los algoritmos desarrollados.

1.3. Alcances y contribuciones

El trabajo de esta memoria considera los siguientes alcances:

- Basado en el trabajo de exploración de alternativas de desarrollo realizado en

ELO307, se evalúan diversas alternativas en base a su utilidad para modelos a escala y se seleccionan los artículos "Duckietown: an Open, Inexpensive and Flexible Platform for Autonomy Education and Research"[16] y "SwiftLane: Towards Fast and Efficient Lane Detection"[17] para el desarrollo de un detector de carriles tradicional y un detector de carriles basado en *deep learning* respectivamente. Por ende, el trabajo se acota a la implementación y evaluación de un algoritmo detector de carriles tradicional y uno basado en *deep learning*.

- El código de ambos algoritmos es implementado en python, utilizando principalmente las librerías TensorFlow[37] y OpenCV[38] en python[39]. Las versiones de librerías y software están especificadas en los repositorios provistos en esta memoria. Los algoritmos son ejecutados en un computador de escritorio (procesador AMD Ryzen 9 5980HS y GPU NVIDIA GEFORCE GTX 1650) y en caso de que al evaluar cualitativamente se considere que el rendimiento es satisfactorio, se ejecuta en un sistema embebido (Jetson TX2). Se utiliza una Jetson TX2 en lugar de la unidad de procesamiento ya disponible en RUPU debido a que en el momento que se desarrolla esta memoria, se planea reemplazar la unidad de procesamiento en los robots de RUPU. Una opción que se encuentra disponible en el Departamento de Electrónica de la Universidad Técnica Federico Santa María es la tarjeta Jetson TX2, la cual posee una unidad de procesamiento gráfico que permitiría obtener un buen rendimiento del detector de carriles basado en *deep learning* relativo a otros sistemas embebidos del mismo costo, en caso de que su rendimiento se considere suficiente en el computador de escritorio.

- Para el entrenamiento del detector de carriles basado en *deep learning* se utiliza el *dataset* TuSimple[40], que corresponde a vídeos grabados en carreteras de Estados Unidos en diferentes condiciones climáticas desde la perspectiva de un conductor. El *dataset* TuSimple es comúnmente utilizado por autores de artículos de investigación para medir el rendimiento de sus detectores de carril basados en *deep learning* debido a que fue uno de los primeros *datasets* en hacerse disponible

públicamente, representado carreteras durante el día y limitando las maniobras del conductor a mantener su carril o cambiar de carril. El utilizar este *dataset* nos permite comparar los resultados de esta memoria con el rendimiento obtenido por otros detectores de carril basados en *deep learning*, pero limita el entrenamiento de la red neuronal a un *dataset* de maniobras de conducción simples en carretera durante el día.

- Para las pruebas experimentales de ambos detectores, se utiliza el *dataset* TuSimple[40] como representación de un ambiente real y vídeos grabados en una pista a escala donde el carril es negro y las líneas delimitadoras son blancas como representación de un ambiente controlado. Los resultados obtenidos de los experimentos se limitan a los contextos anteriormente mencionados y no representan necesariamente el rendimiento de los algoritmos en otras situaciones.
- Se crean visualizaciones de las salidas de ambos algoritmos para que estos sean evaluados cualitativamente y no en base a métricas de precisión. Esto debido a que evidencia[41] reciente muestra que las métricas comúnmente utilizadas para medir el rendimiento de los detectores de carriles no son aptas para tareas orientadas a conducción de vehículos.

Las principales contribuciones de este trabajo se resumen a continuación:

- Desarrollo e implementación de un detector de carriles tradicional y un detector basado en *deep learning*, junto con un repositorio que contiene código e instrucciones para facilitar su instalación.
- Evaluación experimental del rendimiento de ambos algoritmos implementados en diferentes entornos, midiendo latencia y grabando visualizaciones de las predicciones de los detectores, facilitando la comparación de estos. Evaluación realizada en un computador de escritorio y en un sistema embebido en caso de que corresponda.
- Código de demostración que permite la visualización de predicciones de ambos de

algoritmos al mismo tiempo, facilitando a un usuario comprobar y comparar su funcionamiento.

- En línea con los puntos anteriores, se espera que los códigos y evidencia provista en este trabajo contribuyan con directrices para continuar esta línea de trabajo, facilitando la investigación y desarrollo enfocado a conducción autónoma en los modelos a escala desarrollados en el Departamento de Electrónica de la Universidad Técnica Federico Santa María.

1.4. Organización del documento

El resto de los capítulos en este documento siguen la siguiente estructura:

- **Capítulo 2 - Antecedentes:** Se presentan antecedentes y trabajos previos relacionados con el tópico tratado en este trabajo, describiendo brevemente la teoría detrás de los algoritmos utilizados en la literatura reciente.
- **Capítulo 3 - Métodos:** Se describen los detalles teóricos del trabajo realizado, detallando la estructura y funcionamiento del algoritmo detector de carriles tradicional y del algoritmo basado en *deep learning*.
- **Capítulo 4 - Experimentos y resultados:** Detalla brevemente la disposición de los experimentos llevados a cabo, mencionando los datos, el hardware y las métricas utilizadas en la medición. Se presentan los resultados, comparando aquellos obtenidos por el método tradicional con el método basado en *deep learning*.
- **Capítulo 5 - Conclusiones y trabajo futuro:** Se sintetizan las conclusiones obtenidas del análisis de la sección previa y se describen potenciales formas de extender el trabajo realizado.

2. Antecedentes

Este capítulo se aboca a dar una introducción a los algoritmos detectores de carriles tradicionales y aquellos basados en *deep learning*, describiendo la base teórica de estos. Luego de eso, se continua con una reseña de trabajos relacionados con el tópico a tratar en esta memoria, con el fin de entregar una visión general de la literatura actual. En esta reseña se busca describir las técnicas utilizadas en los algoritmos, entregando una distinción entre algoritmos que son clasificados como tradicionales o basados en *deep learning*.

2.1. Base conceptual

2.1.1. Detectores de carril tradicionales

Los detectores tradicionales generalmente resuelven el problema de detección de carril basándose totalmente en datos visibles. El concepto principal de estas técnicas es aprovechar las pistas visibles a través del procesamiento de imágenes utilizando operaciones ajustadas manualmente, como el modelo de colores HSL junto con valores límites para separar píxeles de determinados colores y los algoritmos de extracción de bordes como *Canny*. Cuando los registros visibles no son lo suficientemente robustos, también se utilizan técnicas de pos-procesamiento como Markov, *tracking* o campos aleatorios condicionales. A continuación se describen técnicas y operaciones que usualmente forman parte de un detector de carril tradicional:

HSL & HSV El color de un píxel en una imagen generalmente se representa por sus valores RGB (rojo, verde, azul), pero existen otras representaciones alternativas como HSL (tono, saturación, luminosidad) y HSV (tono, saturación, valor), una representación gráfica de las cuales se puede apreciar en la figura 2.2. En el contexto de diseñar un detector de carril, las representaciones HSL y HSV resultan ser más útiles, ya que se alinean de mayor manera a la forma en que la visión humana percibe colores[42]. En este contexto, un algoritmo se encarga de filtrar los colores de una imagen dependiendo

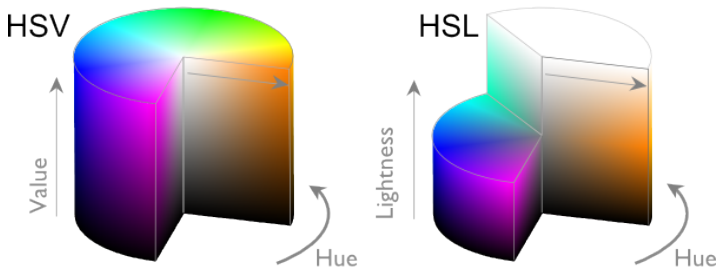


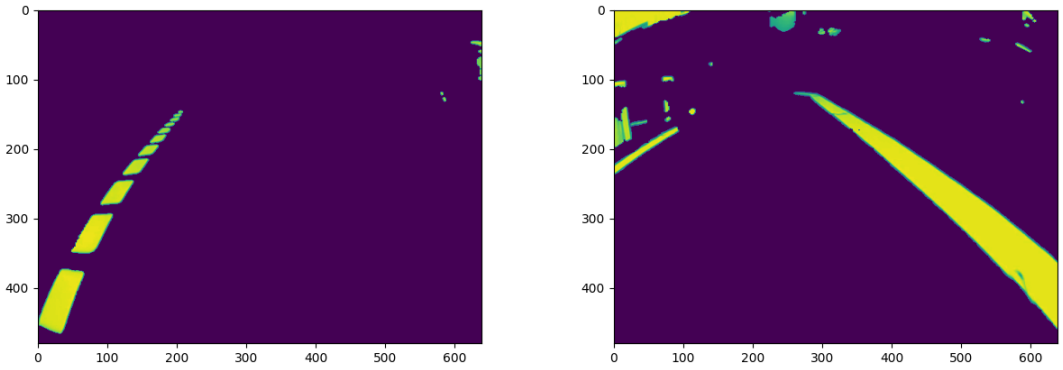
Figura 2.2: Representaciones HSL y HSV. Fuente:[2].

de valores limites que son ajustados manualmente en las dimensiones "H'', "S", "L" o "V". Esto permite a una persona comprender con mayor facilidad que efectos tienen estos valores y ajustarlos con mayor facilidad para lograr su objetivo.

La sigla "H" hace referencia al tono o *hue* en inglés, lo cual representa al pigmento puro presente en el color, como por ejemplo un rojo puro. La sigla "S" indica la saturación del color o en otras palabras, la intensidad del pigmento puro. La sigla "L" en HSL representa la luminosidad, asemejándose a la forma en que diferentes pinturas pueden ser mezcladas para crear un color, específicamente a cuanta pintura blanca o negra se utiliza para generar el color. Mientras que la sigla "V"(valor) de la representación HSV, se comporta de manera similar a un color que es iluminado con luz de diferentes intensidades. La mayor diferencia entre las representaciones HSL y HSV es que un color con el máximo valor de "L" o luminosidad es blanco puro, mientras que un color con máximo "V" o valor, es el color iluminado con luz intensa, como se puede apreciar en la figura 2.2.

Estas representaciones facilitan la segmentación de una imagen por colores específicos en distintas condiciones de luminosidad. Por ejemplo, se utilizan diferentes valores limites en las dimensiones de HSV para filtrar aquellos píxeles de colores blanco y amarillos presentes en la imagen 2.3, y se obtienen las máscaras que se pueden ver en la figura 2.4. Segmentamos por los colores amarillos y blanco y obtenemos las siguientes mascaras como resultado:

Figura 2.3: Imagen antes de ser procesada con HSV. Fuente: [3].



(a) máscara Amarilla **(b)** máscara Blanca

Figura 2.4: Segmentación por colores usando HSV. Fuente: Elaboración propia.

Transformada de Hough El propósito de la Transformada de Hough es identificar líneas rectas en una imagen. Una línea recta puede ser representada por los parámetros (a,b) que representan la pendiente y el intercepto. La línea en coordenadas cartesianas (x,y) se describe con la ecuación 2.1.

$$y = a * x + b \tag{2.1}$$

La línea también puede ser representada en coordenadas polares con los parámetros (ρ, Θ). El primer parámetro, ρ, es la distancia más corta desde el origen hasta la línea (acercándose a la línea perpendicularmente). El segundo, Θ, es el ángulo entre el eje x

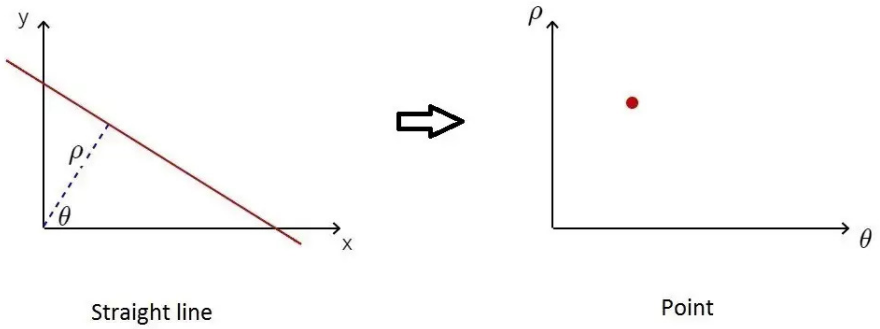


Figura 2.5: línea recta en espacio cartesiano a espacio de Hough. Fuente: [4].

y la línea de distancia. La línea se describe con la ecuación 2.2.

$$\rho = x * cos(\Theta) + y * sin(\Theta) \tag{2.2}$$

Estos parámetros pueden ser representados en otro plano de transformación como se ve en la figura. Este nuevo plano que contiene los parámetros (ρ, Θ), sera llamado espacio de Hough.

Un punto en el plano cartesiano se representa como una sinusoidal en el plano de Hough como se ven la figura 2.5. Esto se demuestra al reemplazar (x,y) por valores constantes en la ecuacion 2.2. Múltiples puntos que forman parte de una misma recta en el espacio cartesiano, forman múltiples sinusoidales en el espacio de Hough, las cuales se intersectan en un punto que corresponde a los valores de (ρ, Θ) que forman la recta a la cual pertenecen los puntos en el plano cartesiano, como se puede ver en la figura 2.6. Esto permite encontrar múltiples puntos que forman una recta en una imagen binaria al encontrar las intersecciones de sus sinusoidales en el plano de Hough. Para aplicaciones reales, el espacio de Hough se divide en cuadriculas con sus respectivos valores de (ρ, Θ) y se implementa un sistema de votos con valores limites, es decir si una sinusoidal pasa por una cuadricula, esta cuadricula obtiene un voto y si una cuadricula obtiene más votos que el valor limite asignado, se considera como una recta presente en la imagen con los valores (ρ, Θ) de tal cuadricula. Una práctica común es utilizar detectores de

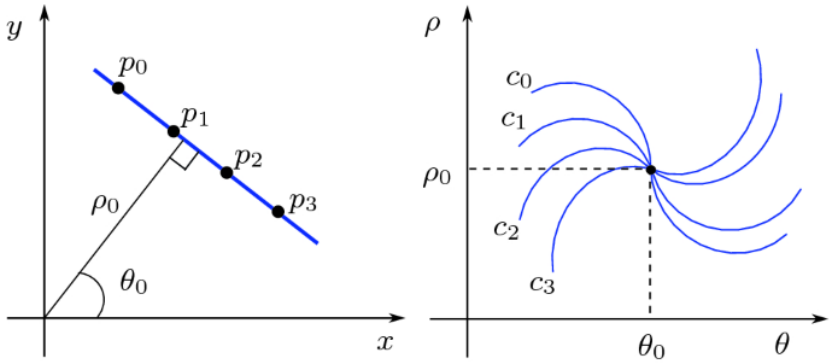
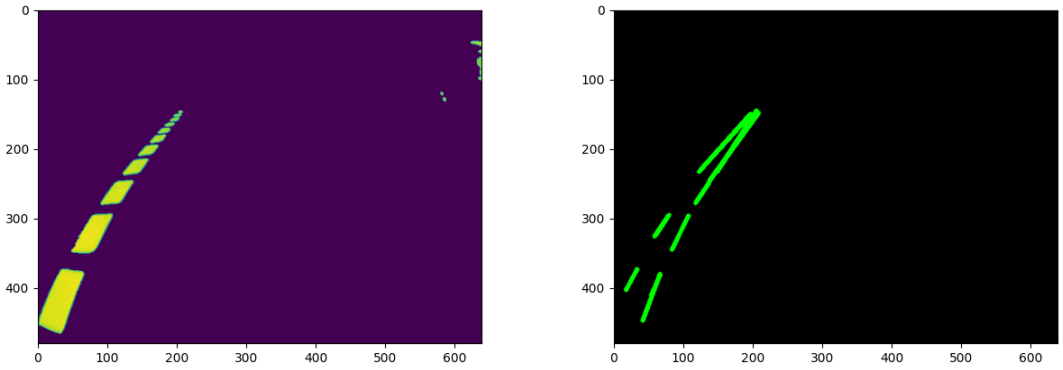


Figura 2.6: Puntos en espacio cartesiano a espacio de Hough. Fuente: [5].



(a) máscara Amarilla **(b)** Transformada de Hough

Figura 2.7: Ejemplo de aplicación de la transformada de Hough. Fuente: Elaboración propia.

bordes como Canny o filtros usando HSV/HSL para generar una máscara binaria y luego utilizar la transformada de Hough para detectar líneas rectas dentro de la imagen, como se puede ver en la figura 2.7.

2.1.2. Detectores de carril basados en *deep learning*

Los detectores de carril basados en *deep learning* utilizan redes neuronales profundas, las cuales consisten en una red neuronal artificial que posee múltiples capas ocultas entre su salida y su entrada. Es esta característica la que les otorga la denominación de "profunda", y el principal propósito de tener una gran cantidad de capas es el facilitar el aprendizaje de representaciones de alto nivel de la información con la cual la red es

entrenada. Las redes neuronales pueden ser divididas en categorías dependiendo de su estructura y en cómo son entrenadas.

Redes convolucionales Una red neuronal convolucional (CNN) es una clase de red neuronal que toma como entrada un tensor y extrae información relevante mediante la aplicación de una serie de bancos de filtros convolucionales. Usualmente este tensor representa una imagen a color, pero las CNNs pueden utilizarse para procesar cualquier tipo de información que tenga algún tipo de organización espacio-temporal. Este tipo de redes también suele incorporar operaciones de submuestreo espacial, ya sea incorporado en las mismas convoluciones o por medio de capas específicas para ese propósito, denominadas capas de "pooling", la cuales normalmente funcionan subdividiendo la imagen de entrada en varias partes y tomando el valor máximo o promedio de cada parte. Las CNNs pueden considerarse una versión con arquitectura restringida de los perceptrones multicapa, también conocidos como redes completamente conectadas (del inglés Fully-Connected o FC). De manera similar a como las primeras redes neuronales inspiraron su arquitectura en la organización de las neuronas dentro del cerebro, las CNNs se inspiran en el funcionamiento de la corteza visual, la cual organiza a sus neuronas asignando a cada una un campo receptivo limitado, es decir, solo una pequeña porción de la imagen. Considerando el conjunto completo de neuronas que perciben la imagen, el conjunto de campos receptivos procesa el estímulo visual de manera análoga a como lo hace un filtro convolucional de dos dimensiones. Comparadas con las redes completamente conectadas, las redes convolucionales resultan ser mucho más eficientes para el procesamiento de imágenes [43]. Cada convolución puede caracterizarse a través de un kernel o filtro, el cual se asimila a una ventana móvil que se desplaza a lo largo de la entrada, multiplicando y sumando en cada paso, como se muestra en la figura 2.8. Teniendo una matriz de entrada X de tamaño (m x n), siendo X_{ij} un valor perteneciente a la matriz X para $\{\text{i } \epsilon \mathbb{N}\text{: } 1 \leq i \leq m\}$ y $\{\text{j } \epsilon \mathbb{N}\text{: } 1 \leq i \leq n\}$, y considerando un kernel H como una matriz (l x l), siendo l comúnmente un numero natural de valores pequeños como 3, 5, etc. El resultado de la convolución de la matriz de entrada X con el kernel

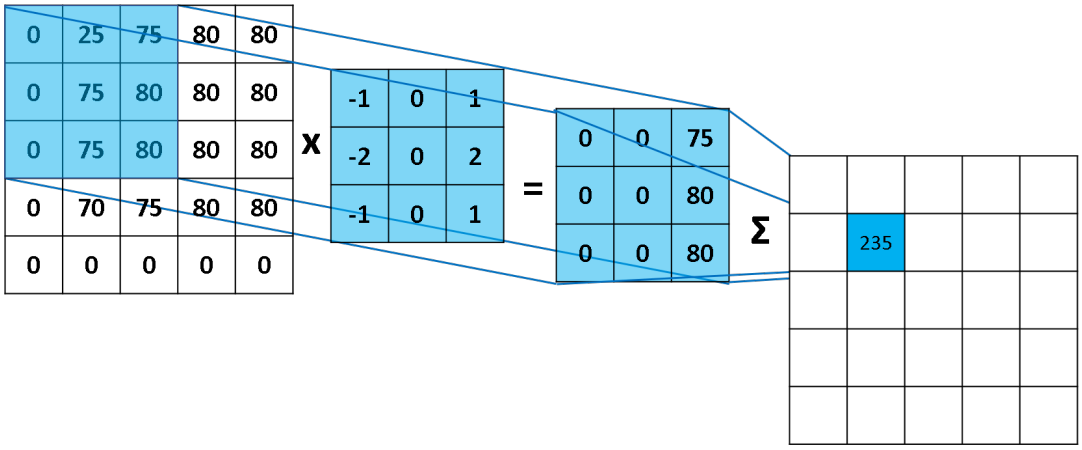


Figura 2.8: Un paso en el proceso de convolución. Visualizando la multiplicación del kernel con parte de la entrada y posterior suma. Fuente: [6].

H es otra matriz Y, la cual esta compuesta por los valores Y_{ij} que están dados por la ecuación 2.3.

$$Y_{ij} = \sum_{u=1}^{l} \sum_{s}^{l} H_{us} * X_{a(i,u),b(j,s)} \tag{2.3}$$

Las funciones a(i,u) y b(j,s) son funciones de índice que varían dependiendo de los parámetros de "stride" y "padding" de la capa convolucional. El parámetro "stride" indica el tamaño de los salto con los que el kernel de convolución se desliza sobre la imagen, mientras que "padding" hace referencia a cuánto relleno de ceros debe colocarse alrededor de la imagen de entrada. Considerando los parámetros "stride" = 1 y "padding" = 0, las funciones de índice serian $a(i,u) = i + u$ y $b(j,s) = j + s$.

La salida de cada capa convolucional es llamada "mapa de características" o "feature map" en inglés, estos mapas de características extraen información acerca de la imagen como bordes o patrones relevantes para cumplir el objetivo para el cual se ha entrenado la red. Diferentes filtros extraen diferentes mapas de características como se muestra en la figura 2.9, estos mapas pueden ser procesados por capas posteriores como una capa de "pooling" para disminuir la dimensionalidad espacial a costo de perdida de información, para luego entregar esta salida a una capa completamente conectada.

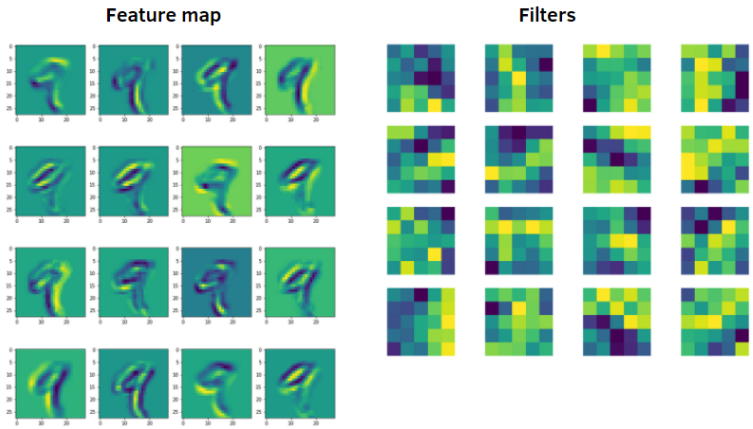


Figura 2.9: Diferentes mapas de características obtenidos a partir de diferentes filtros de convolución aplicados sobre una imagen de un 9 del *dataset* MNIST[7]. Fuente: [8].

Una estructura genérica de CNN consiste en una capa de entrada y de salida, conectadas entre sí por múltiples capas intermedias, junto con capas de "pooling" como se muestra en la figura 2.10. La capa de entrada se encarga de recibir la información en el formato correspondiente y ponderarla para las capas posteriores, mientras que la capa de salida entrega el resultado de la red según lo requerido, sea este una imagen, categoría, o segmentación de lo observado. Las capas intermedias se componen en su mayoría de filtros convolucionales, en donde cada uno se encuentra conectado solo a una región de la salida de la capa anterior y aplica la operación de convolución correspondiente sobre esta. Los parámetros de estos filtros son optimizados durante la etapa de entrenamiento, en la cual la red neuronal aprende a reconocer patrones a partir de los datos provistos. Esto se logra al comparar la imagen de salida de la red neuronal con la imagen objetivo, calculando la diferencia entre ambas mediante la función de pérdida. Es posible obtener el gradiente que minimiza resultado de la función de perdida en función de los valores de los pesos que conectan las neuronas de la red, luego este gradiente puede ser utilizado para actualizar los parámetros internos de la red neuronal mediante "backward propagation" a lo largo de múltiples iteraciones. Al culminar el entrenamiento de la red neuronal luego de un número de iteraciones, se espera que los parámetros se encuentran

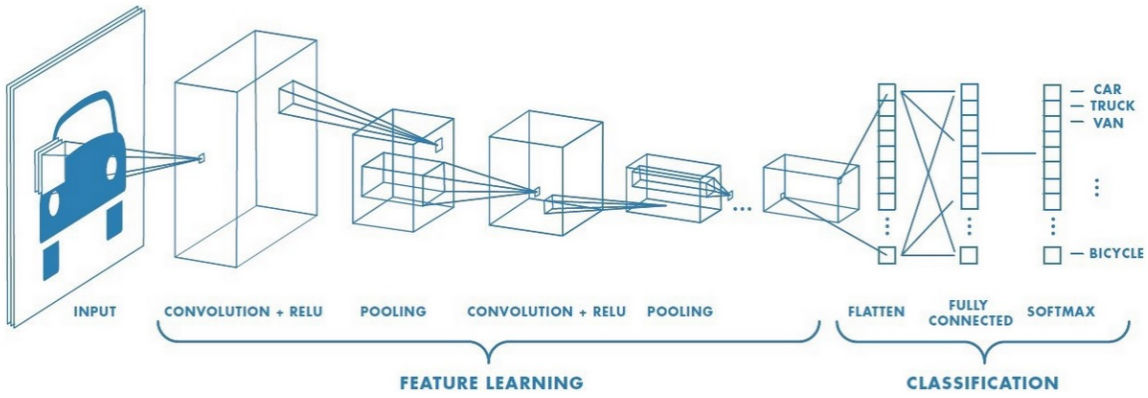


Figura 2.10: Representación de una CNN genérica. En este caso, la red procesa una imagen de entrada a través de sus capas para clasificar el tipo de imagen que se le presenta. Las CNNs de clasificación están entre las redes más estudiadas, y por lo mismo son el tipo de red utilizada en este trabajo. Fuente: [9].

ajustados a lo que se puede referir como sus valores óptimos, o por lo menos cercanos a estos. Es importante mencionar que el correcto entrenamiento de la red depende de la correcta selección de múltiples parámetros e hiperparámetros, para lo cual no hay una forma sistemática de selección y dependen en gran parte de técnicas heurísticas en conjunto con un proceso de prueba y error, por lo que el uso correcto de métricas de rendimiento es esencial para el desarrollo de una red neuronal eficaz. Finalmente, una vez entrenada la red neuronal, se lleva a cabo el proceso llamado inferencia, en donde la red neuronal se mantiene fija y se le dan las entradas para solamente calcular la salida correspondiente, ya sea para evaluar su desempeño ante diferentes datos o utilizar la salida de esta.

Es posible aumentar la cantidad de capas internas en la red con el fin de mejorar la precisión de la red, pero esto conlleva dificultades adicionales, como un aumento en el costo computacional y problema de desvanecimiento de gradiente[44], este problema ocurre al agregar múltiples capaz con determinadas funciones de activaciones que producen que el valor del gradiente calculado durante el entrenamiento sea muy cercano a 0 y no permita actualizar correctamente el valor de los pesos de la red. Para combatir este problema se han desarrollado nuevas alternativas como las redes residuales

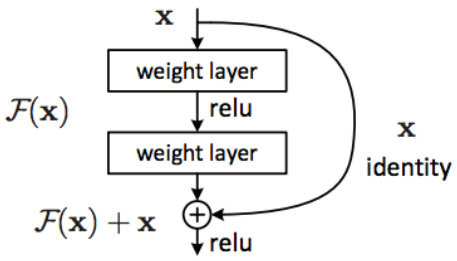


Figura 2.11: Bloque residual. Fuente: [10].

(ResNet)[45].

Redes residuales Evidencia[46, 47] revela que la profundidad de las redes neuronales artificiales es de crucial importancia para problemas de reconocimiento visual, y los resultados lideres[47–50] en el dataset ImageNet[51] hacen uso de modelos "profundos" con una profundidad de 16 a 30 capas. Pero como se mencionó anteriormente, una mayor profundidad conlleva otras dificultades para el entrenamiento de la red, como el problema de desvanecimiento de gradiente[44] y el problema de degradación en el cual al aumentar la profundidad de la red, la precisión se satura y luego se degrada rápidamente. Para reducir los efectos de estos problemas las redes residuales implementan "bloques residuales" como se puede observar en la figura 2.11.

Formalmente, denotando el mapeo subyacente deseado como $H(x)$, dejamos que las capas no lineales apiladas se ajusten a otro mapeo de $F(x) = H(x) - x$. El mapeo original se reasigna en $H(x) = F(x) - x$. Esta modificación en la que se agrega la suma de x, también es conocida como una "conexión salto". Las conexiones de salto en ResNet resuelven el problema de la desaparición del gradiente en las redes neuronales profundas al permitir que fluya esta ruta de atajo alternativa para que fluya el gradiente. La otra forma en que estas conexiones ayudan es al permitir que el modelo aprenda las funciones de identidad, lo que garantiza que la capa superior funcionará al menos tan bien como la capa inferior, y no peor. En el mejor de los casos, las capas adicionales de la red neuronal profunda pueden aproximar mejor el mapeo de entrada a la salida que la contra parte menos profunda y reduce el error por un margen significativo. Por

lo tanto, se espera que ResNet funcione igual o mejor que las simples redes neuronales profundas.

Entrenamiento Para entrenar una red neuronal, normalmente se usa un conjunto de datos etiquetados con la salida esperada para cada ejemplo, lo que se denomina aprendizaje supervisado, esto entra en contraste con el aprendizaje no supervisado, donde algoritmos intentan "aprender" patrones o estructuras a partir de datos, sin entregarles una salida esperada de forma explícita, sino que simplemente deben minimizar una función de costo asignada (ej. dividir un conjunto de puntos en subgrupos cohesionados, intentando minimizar la distancia de cada punto al promedio de su respectivo subgrupo). En aprendizaje supervisado, la forma más común de ajustar los pesos durante el entrenamiento es usando métodos basados en gradiente, donde se define una función de costo a minimizar, la cual asigna un costo mayor para la red mientras la salida obtenida sea más distinta de lo esperado, mediante el algoritmo de "backward propagation" se calcula el gradiente de la función de costo respecto a cada uno de los pesos de la red para luego actualizar los pesos en una dirección que reduzca el costo total. Algunos de los principales parámetros del proceso de entrenamiento, también llamados hiperparámetros, son la tasa de aprendizaje que define cuánto se modifican los pesos por cada paso del entrenamiento, y la cantidad de épocas, que corresponden al número de iteraciones sobre el conjunto completo de datos de entrenamiento. Tanto al momento de entrenar como para hacer inferencia con un modelo de red, es común tomar un conjunto pequeño de imágenes de entrada, llamado lote o "batch", para evaluarlo en la red de forma simultánea, lo que permite aplicar optimizaciones a nivel de código para acelerar el procesamiento de la red. Todo el proceso de entrenamiento y definición de la arquitectura de red se suele programar en "frameworks" especializados en redes neuronales, dentro de los cuales los más utilizados en la actualidad son PyTorch y TensorFlow.

Es una práctica común el aplicar técnicas de regularización al momento de entrenar la red, que tienen como propósito evitar el sobreajuste (overfitting) a los datos de

entrenamiento y así poder mejorar la generalización del modelo. Existen varias formas de regularización, algunas agregan explícitamente un término adicional a la función de costo, el cual es proporcional a la norma L1 o L2 del conjunto de pesos de la red, otras son más implícitas, como la normalización por lotes, o "batch normalization" (BN), la cual ha sido adoptada por la mayoría de las CNN modernas como un enfoque estándar para lograr una convergencia rápida y un mejor rendimiento de generalización.

Transfer learning Es una técnica de aprendizaje de máquinas en la cual se entrena un modelo con el objetivo de extraer información de uno o más tareas de origen, para luego reutilizar partes de este modelo en una nueva tarea objetivo[52]. Esta herramienta es utilizada comúnmente en el campo de visión por computador, entrenando redes neuronales convolucionales profundas preentrenadas en el dataset ImageNet[51], como una ResNet, para ser reutilizadas en un modelo posterior. Modelos preentrenados están disponibles en los *frameworks* como PyTorch, Tensorflow o se pueden descargar de repositorios públicos de Git.

El modelo preentrenado utilizado para el aprendizaje por transferencia suele denominarse como *backbone*. Durante el procedimiento de adaptación, es común reemplazar las capas superiores del modelo original, también conocidas como *classifier head*, con nuevas capas enfocadas al cumplimiento de la nueva tarea. Esto debido a que las primeras capas del *backbone* contienen características más genéricas (por ejemplo, detectores de bordes o colores). Por lo tanto, el nuevo modelo es a menudo entrenado con las capas inferiores congeladas. Este enfoque es común si el conjunto de datos no tiene muchas muestras para evitar el sobre ajuste.

2.2. Trabajos relacionados

Para el desarrollo de esta memoria se exploraron múltiples alternativas de diseño de detectores de carril con una cámara tradicional. A continuación, se mencionan algunas de las alternativas relevantes que logran relativamente buena precisión y baja latencia, pe-

ro representan paradigmas de diseño diferentes a los seleccionados en esta memoria para solucionar el problema de detección de carril. "Combining Statistical Hough Transform and Particle Filter for Robust Lane Detection and Tracking"[12] presenta un detector de carriles tradicional que utiliza "Inverse Perspective Mapping", una técnica comúnmente utilizada en otros detectores de carril para remover efectos de perspectiva. Los detectores de carril basados en *deep learning* pueden ser clasificados dependiendo de los métodos que utilizan para detectar carriles: basados en segmentación, basado en anclas, clasificadores por filas y basados en predicción paramétrica. "Rethinking Efficient Lane Detection via Curve Modeling"[13], "Towards End-to-End Lane Detection: an Instance Segmentation Approach" y "Keep your Eyes on the Lane: Real-time Attention-guided Lane Detection" presentan detectores basados en *deep learning* que utilizan los métodos basados en predicción paramétrica, segmentación y en anclas respectivamente.

2.2.1. "Combining Statistical Hough Transform and Particle Filter for Robust Lane Detection and Tracking"

Los autores de este articulo[12] proponen un algoritmo detector de carriles que utiliza 4 operaciones distintas. Primero se utiliza "Inverse Perspective Mapping" (IPM), para remover el efecto de perspectiva basándose en una hipótesis de que el terreno es plano. Este algoritmo entrega como salida una transformación de imagen vista desde arriba.

Se asume que las marcas delimitadoras del carril cercanas al vehículo aparecen como líneas rectas, las cuales son definidas por la ecuación 2.4.

$$p = x * cos(\theta) + y * sen(\theta) \tag{2.4}$$

Donde x e y corresponden a las coordenadas en los ejes horizontal y vertical respectivamente, mientras que p y θ son parámetros de la línea.

Luego se utiliza la Transformada de Hough Estadística (SHT)[53], que a diferencia de la Transformada de Hough, es un modelo no supervisado que modela explícitamente las

dependencias entre variables y por ende no depende de la elección de origen del sistema de coordenadas en la imagen. Este algoritmo codifica bien el contenido de alineación de las imágenes y es resistente al ruido, pero es más costoso computacionalmente que la transformada estándar de Hough porque se utilizan todos los píxeles de la imagen (en lugar de seleccionar solo los bordes), además las colas de los núcleos deben tenerse en cuenta en la computación de las estimaciones.

Posteriormente se agregan restricciones adicionales para especificar cuales de las líneas obtenidas de SHT corresponden realmente a delimitadores de carril:

- Los delimitadores de carril son paralelos entre si, por lo que los ángulos θ deben ser similares
- Los anchos de los carriles son similares, por lo que las distancias entre delimitadores de carril deben similares.

Dado al alto costo computacional de SHT, luego de su uso inicial se utiliza un filtro de partículas para rastrear los delimitadores de carriles detectados y actualizar los parámetros del modelo de carril, lo que reduce considerablemente el tiempo de cálculo.

2.3. "Rethinking Efficient Lane Detection via Curve Modeling"

En este articulo[13] se proponen un algoritmo de *deep Learning* llamado BézierLaneNet basado en la utilización de la curva de Bezier paramétrica debido a su facilidad de cálculo, estabilidad y altos grados de libertad de transformaciones. El modelo tiene la siguiente arquitectura:

Se utiliza como *backbone* un enconder como ResNet, específicamente las características de la tercera capa, reforzado por "feature flip fusion" para agregar características de carriles opuestos, seguido por un "pooling" hacia una dimensión y dos capaz de redes convolucionales de una dimensión. Finalmente, la red predice curvas de Bezier a través

de una rama de clasificación y una rama de regresión.

Los autores proclaman que el modelo consigue un nuevo rendimiento de estado del arte bajo el "benchmark" LLAMAS. Con precisión favorable en los datasets CULane y TuSimple, mientras mantiene baja latencia "(>150 FPS)" y un modelo pequeño "(<10 millones de parámetros)".

2.4. "Keep your Eyes on the Lane: Real-time Attention-guided Lane Detection"

Este articulo[14] propone un modelo de "Deep Learning" de una etapa que utiliza el método de anclas para detectar carriles. La arquitectura del modelo se muestra en la siguiente imagen:

El modelo recibe como entrada una imagen RGB y la salida son las líneas que delimitan los carriles. Para generar estas salidas se utiliza primero una red convolucional como "backbone", la cual genera un mapa de características que luego pasa por una capa de "pooling" para extraer las características de cada ancla. Luego estas características son combinadas con un set de características globales producidas por un módulo de atención. Al combinar las características, el modelo puede usar información de otros carriles con mayor facilidad, lo cual puede ser necesario en casos con condiciones como oclusión ambiental o la falta de marcas de carril visibles. Finalmente, las características combinadas entran a una capa completamente conectada para predecir las líneas de carril finales.

Los autores mencionan que al momento de la publicación de este artículo, en 2020, el modelo propuesto supero a otros modelos de estado del arte en cuanto a eficacia y precisión. El computador utilizado para las pruebas contiene un procesador Intel i9-9900KS y una GPU RTX 2080 Ti.

2.5. "Towards End-to-End Lane Detection: an Instance Segmentation Approach"

Los autores del articulo[15] proponen un modelo de "Deep Learning" que a diferencia de algunos otros modelos, no predice un numero predeterminado de carriles si no que resuelve el problema de detección de carril como un problema de detección de instancias donde cada carril forma una instancia. La arquitectura del modelo propuesto es la siguiente: Se utiliza un encoder que recibe la imagen inicial y su salida es compartida por dos ramas, una de rama de segmentación y otra de "embedding". La rama de "embedding" genera un "embedding" N-dimensional por pixel del carril, para que "embeddings" del mismo carril estén más cerca. La rama de segmentación se utiliza para generar una máscara binaria con la cual se eliminan los pixeles de fondo del resultado de la rama de "embedding", obteniéndose solo los "embeddings" del carril, los cuales son agrupados en "clusters", siendo así la salida del modelo una colección de píxeles por carril.

Los autores mencionan que el modelo es capaz de funcionar a 50 FPS y detectar un numero variable de carriles, además de soportar maniobras de cambio de carril, a diferencia de otros modelos. No se especifica el hardware utilizado para las pruebas.

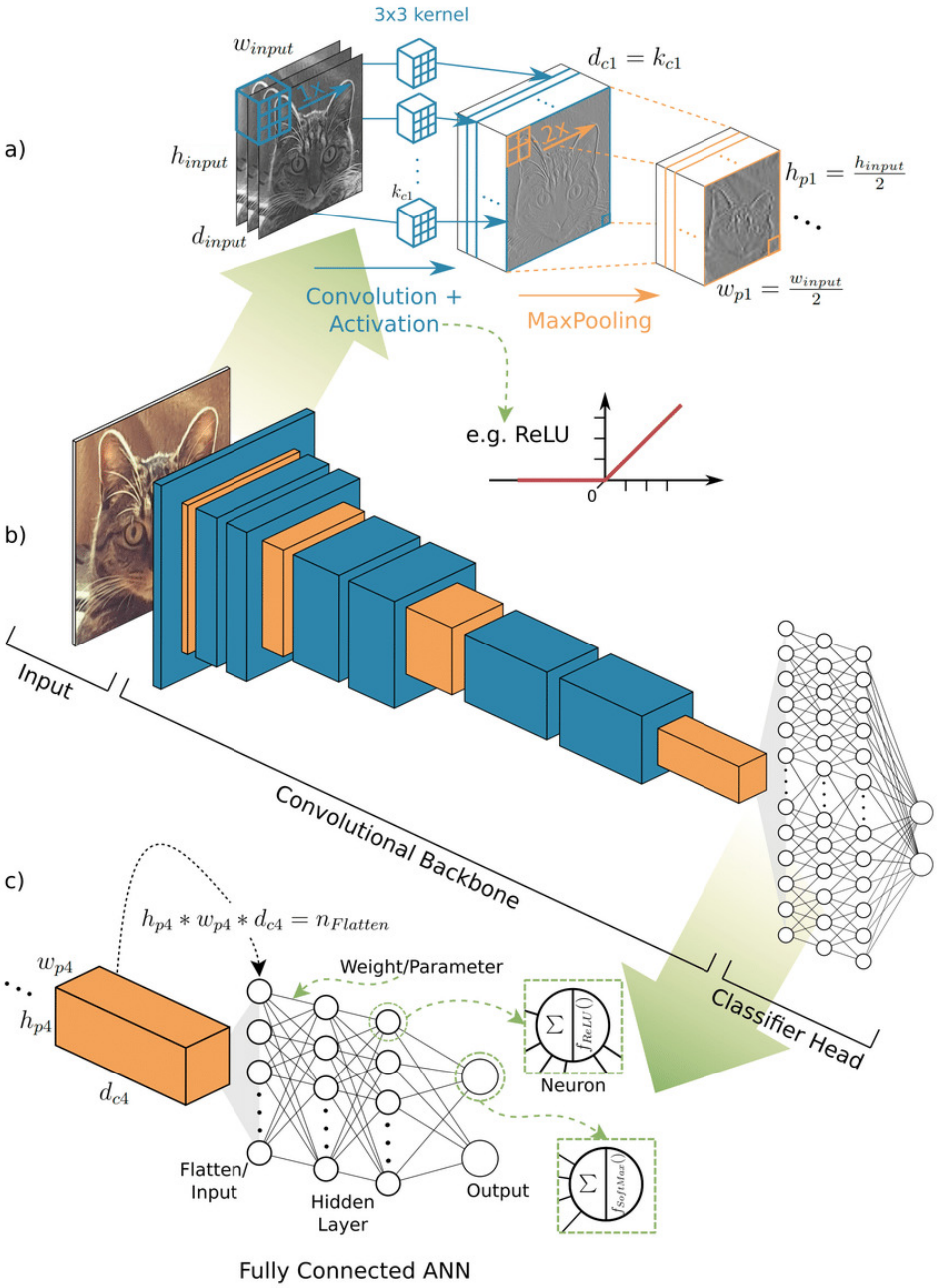
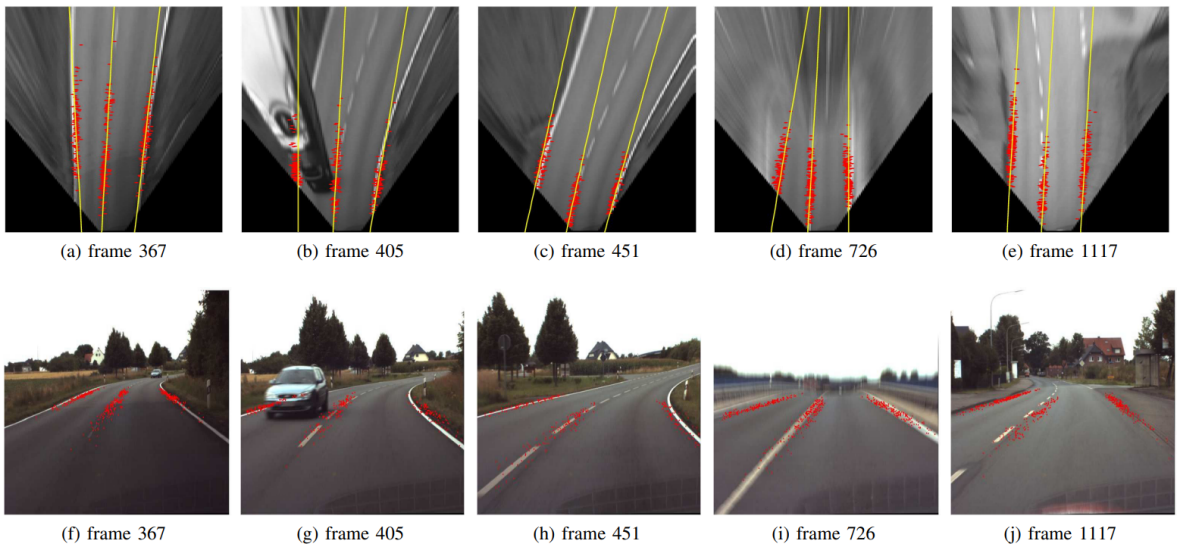


Figura 2.12: Ejemplo de CNN profunda genérica con *backbone* y *classifier head* identificados. Fuente: [11].



(a) frame 367 (b) frame 405 (c) frame 451 (d) frame 726 (e) frame 1117

(f) frame 367 (g) frame 405 (h) frame 451 (i) frame 726 (j) frame 1117

Figura 2.13: Resultados del algoritmo en imágenes consecutivas. Las superiores e inferiores corresponde a la imagen IPM y la imagen original con las partículas proyectadas. Fuente: [12].

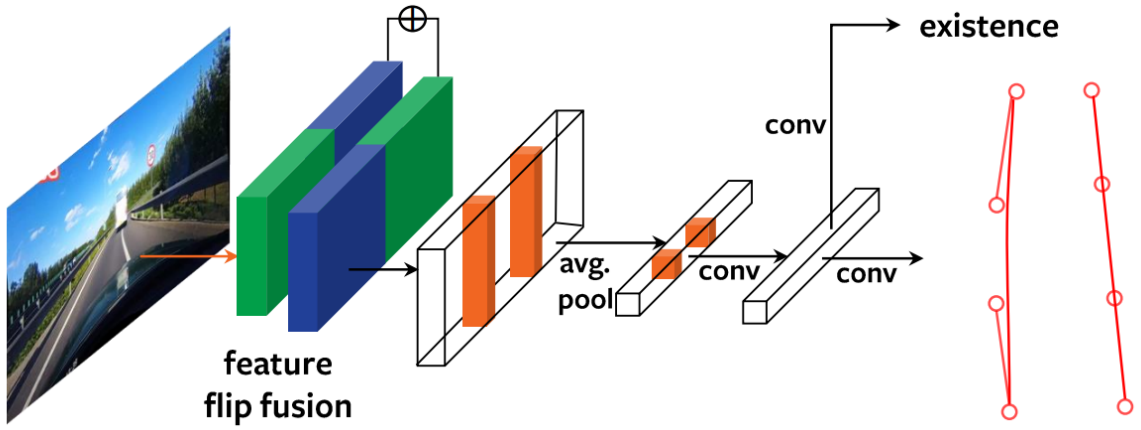


Figura 2.14: Arquitectura de BézierLaneNet. Fuente: [13].

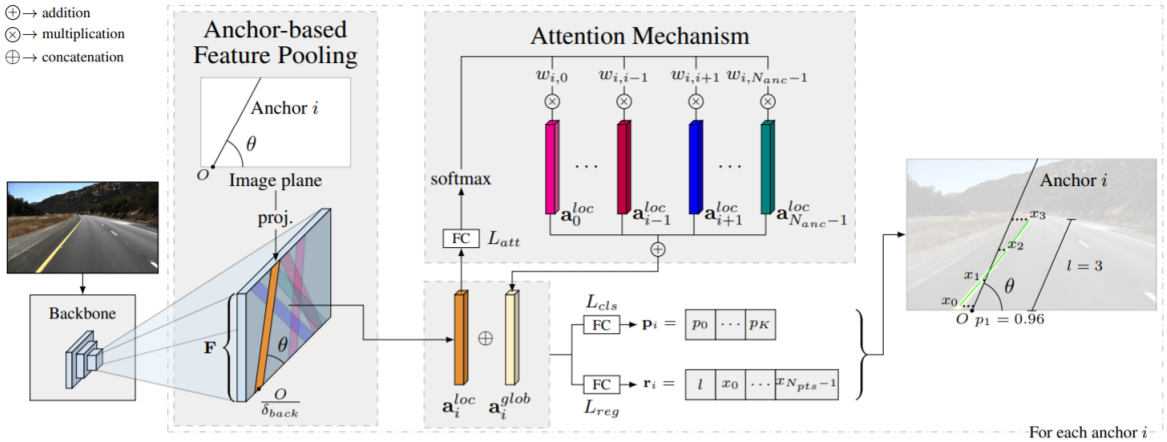


Figura 2.15: Detector de carriles LaneATT. Fuente: [14].

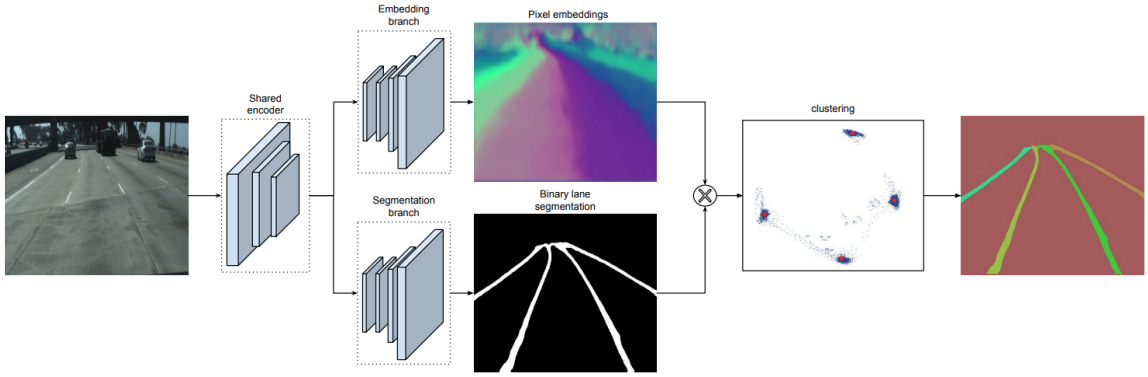


Figura 2.16: Arquitectura de LaneNet. Fuente: [15].

3. Métodos

Este capítulo presenta una descripción de los algoritmos detectores de carriles seleccionados a implementar y evaluar. Esta descripción es complementada por los códigos desarrollados en python que se encuentran disponible en los repositorios[54, 55] que acompañan a esta memoria. Los algoritmos implementados están basados en los siguientes artículos:

- Detector tradicional: "Duckietown: an Open, Inexpensive and Flexible Platform for Autonomy Education and Research"[16].
- Detector basado en *deep learning*: "SwiftLane: Towards Fast and Efficient Lane Detection"[17].

Se presentan aspectos generales sobre su diseño y otros aspectos relevantes que se consideraron al realizar su implementación en código.

3.1. Detector tradicional: "Duckietown: an Open, Inexpensive and Flexible Platform for Autonomy Education and Research"

"Duckietown" es una plataforma a escala diseñada para investigación y educación en el área de vehículos autónomos. Esta plataforma ofrece una amplia gama de funcionalidades a un bajo costo. Los robots de la plataforma perciben el mundo con una sola cámara monocular y realizan todo el procesamiento a bordo con una Raspberry Pi 2, pero son capaces de: seguir carriles evitando obstáculos, peatones a escala y otros robots. Para lograr estos objetivos utilizando solo una cámara monocular y una Rapsberry Pi 2 con una limitada capacidad de cómputo, se propone un detector de carriles tradicional.

El detector primero reduce la resolución de la imagen para disminuir la cantidad de datos a procesar, luego se utilizan dos operaciones sobre la imagen en paralelo, se aplica un detector de bordes Canny para filtrar todos los bordes de objetos presentes en la imagen,

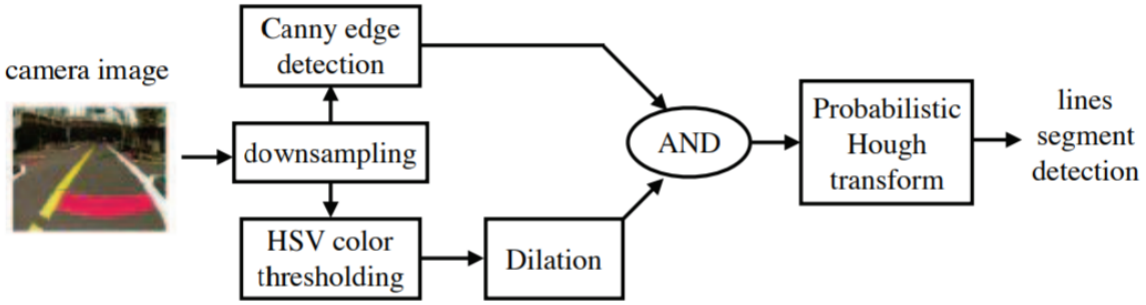


Figura 3.17: Detector de carriles tradicional implementado por Duckietown. Fuente: [16].

y también se utilizan los valores HSV de los píxeles en la imagen en conjunto con valores HSV límites de colores objetivos para filtrar objetos con estos colores. Luego se aplica una operación *AND* entre los resultados de estos filtros y finalmente se obtienen los segmentos individuales utilizando una transformada de Hough probabilística. La figura 3.17 muestra el proceso mencionado anteriormente.

El código del detector elaborado en esta memoria[54], fue implementado en base a los códigos puestos a disposición por los autores del articulo[56]. Las operaciones que involucran HSV, el detector de bordes Canny y la transformada de Hough son implementadas con la librería OpenCV[38].

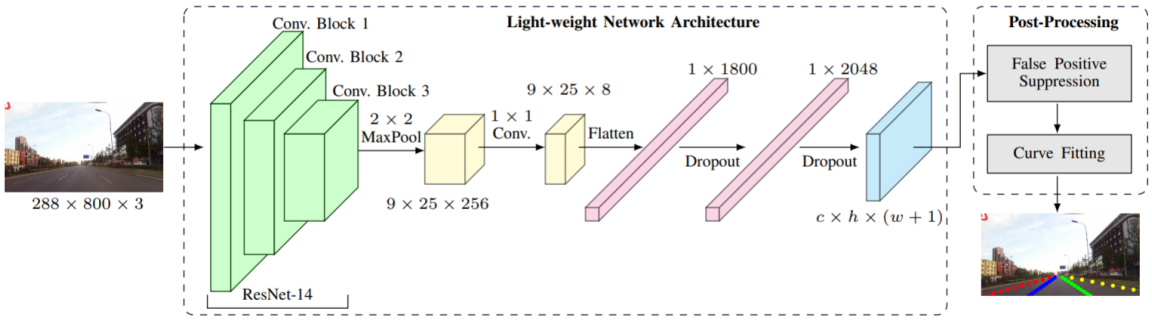


Figura 3.18: Arquitectura de modelo "Swiftlane". Fuente: [17].

3.2. Detector basado en *deep learning*: "SwiftLane: Towards Fast and Efficient Lane Detection"

3.2.1. Arquitectura

"Swiftlane" es un modelo detector de línea basado en *deep learning* con una arquitectura compacta en comparación a otras redes neuronales profundas, que ha sido diseñada para ser utilizada con sistemas embebidos al lograr uno de los mayores *throughput* de los modelos del estado del arte. La arquitectura de "Swiftlane" se muestra en la figura 3.18. Este modelo utiliza como *backbone* una ResNet-14, es decir de 14 capas de profundidad, la cual corresponde a una ResNet-18 como se ve en la figura 3.19, preentrenada en el dataset ImageNet y a la cual se le eliminan las ultimas 4 capas convolucionales.

ImageNet es un dataset que contiene 14.197.122 imágenes anotadas según la jerarquía de WordNet, como las que se pueden apreciar en la figura 3.20. Es comúnmente utilizado en clasificación de imágenes y detección de objetos. Al preentrenar una red ResNet-18 con ImageNet se espera que los filtros ya entrenados sean capaces de obtener mapas de características relevantes para la tarea de detección de carriles.

Se eliminan las ultimas 4 capas del modelo ResNet-18, con el fin de hacer la red mas compacta, disminuyendo la complejidad computacional y se espera que los mapas de características obtenidos como salida de la capa 14 sean suficientes para la tarea de detección de carriles. Seguido al *backbone* se agrega una capa de *max pooling* de 2x2 y

una capa convolucional 1x1 que son utilizadas para reducir la dimensionalidad espacial y el número de canales, para luego entregar su salida al *classifier head* que corresponde a dos capas completamente conectadas entrenadas con *dropout* para disminuir las posibilidades de *overfitting*.

Swiftlane utiliza el método de clasificación por filas, este tipo de métodos predice la salida en forma de una grilla con menor dimensión que la resolución de la imagen original, entregando un valor por fila en la cuadricula por línea limitadora, el uso de esta grilla permite que los métodos de detección por filas tengan menor costo computacional comparado a los comúnmente utilizados métodos basados en segmentación, que generalmente tienen un énfasis en obtener una clasificación precisa por píxel en lugar de especificar la forma de la línea.

La comparación entre ambos métodos se visualiza en la figura 3.21, supongamos que el tamaño de la imagen es H × W. En general, el número de anclajes de fila predefinidos y el tamaño de cuadrícula son mucho menores que el tamaño de una imagen, es decir, h<H y w<W. De esta forma, la formulación de segmentación necesita realizar clasificaciones H × W que sean (C + 1)-dimensional, mientras que el método de clasificación por filas solo necesita resolver problemas de clasificación C × h que son (w + 1)-dimensionales. De esta manera, la escala de cálculo se puede reducir a C×h×(w+1) mientras que el de la segmentación es H × W × (C + 1).

En el caso de Swiftlane, la región de la imagen que contiene carriles se divide en un número predefinido de anclajes de fila (h) y cada anclaje de fila se divide en un número predefinido de celdas de cuadrícula (w) como se muestra en la figura 3.22. El número de carriles (c) está predefinido, y para cada carril, las ubicaciones de los carriles están representadas por una cuadrícula de h × ancho. Se adjunta una celda adicional al final de cada ancla de fila para indicar la ausencia de un carril en particular en esa ancla de fila.

El tensor de salida representa la puntuación de cada celda de cuadrícula (incluida la

celda sin carril) perteneciente a cada carril en cada ancla de fila. $S_{i,j,k}$ representa la puntuación de la celda de rejilla k-ésima en el ancla de fila j-ésima que pertenece al carril i-ésimo que se puede obtener mediante la ecuación 3.5.

$$S_{i,j,k} = \ f(X),\ s.t.\ i\ \epsilon\ [1,h],\ k\ \epsilon\ [1,w+1] \tag{3.5}$$

Aquí, f, X, c, h y w representan el modelo de clasificación, la imagen de entrada, el número de carriles, el número de anclas de fila y el número de celdas de cuadrícula, respectivamente. Luego, los puntos de los carriles se pueden extraer eligiendo la celda de cuadrícula con la puntuación más alta en cada ancla de fila para cada carril. Si la última celda de cuadrícula no es la celda con la puntuación más alta, la ubicación del i-ésimo carril en la j-ésima fila el anclaje viene dado por la ecuación 3.6.

$$Loc_{i,j} = argmax_k(S_{i,j,k}),\ s.t.\ k\ \epsilon\ [1,w] \tag{3.6}$$

Tener la puntuación más alta en la última celda de cuadrícula implica que el carril considerado no está presente en el ancla de fila seleccionada.

3.3. Función de perdida

En el artículo original se define la pérdida de clasificación como la pérdida logarítmica de verosimilitud negativa que viene dada por la ecuación 3.7.

$$L_{cls} = \sum_{i=1}^{C}\sum_{j=1}^{h} -\alpha_{i,j,T_{i,j}} * log(P_{i,j,T_{i,j}}) \tag{3.7}$$

Aquí, Ti,j denota la ubicación correcta (celda de grilla) del i-ésimo carril en el ancla de la j-ésima fila según la realidad del terreno y Pi,j,k denota la probabilidad de la k-ésima celda de grilla en el ancla de la j-ésima fila perteneciente al i-ésimo carril que se puede

obtener mediante la ecuación 3.8

$$P_{i,j,k} = softmax(S_{i,j,k}) \tag{3.8}$$

En la ecuación 3.7, $\alpha_{i,j,k}$ es el factor de modulación para el ajuste de pérdida focal descrito por la ecuación 3.9, como se menciona en [57].

$$\alpha_{i,j,k} = (1 - P_{i,j,k})^{\gamma} \tag{3.9}$$

Pero durante el transcurso de esta memoria no se encontraron mejoras significativas en el rendimiento de la red durante el entrenamiento al utilizar esta función de perdida, por lo que fue reemplazada con la función de perdida utilizada en[20] que corresponde a la entropía cruzada categórica a lo largo de las filas en la grilla de salida, cuyos detalles de implementación están disponibles en la documentación de Tensorflow[58] y es expresada matemáticamente como la ecuación 3.10.

$$L_{cls} = \sum_{i=1}^{C} \sum_{j=1}^{h} L_{CE}(P_{i,j,:}, Y_{i,j,:}) \tag{3.10}$$

Donde L_{CE} corresponde a la entropía cruzada, $P_{i,j,:}$ es el vector dimensional (w + 1) que representa la probabilidad de seleccionar (w + 1) celdas de cuadrícula para el ancla de fila j-ésima de carril i-ésimo, $Y_{i,j,:}$ es la etiqueta *one-hot* de las ubicaciones correctas.

3.3.1. Optimizador y parámetros de entrenamiento

El algoritmo de optimización utilizado para entrenar el modelo es descenso gradiente estocástico con momento[59] con una tasa de aprendizaje inicial de 0,1, momento de 0,9 y una caída de pesos de $1 * 10^{-4}$ agregada para disminuir el *overfitting*. El modelo se entrena durante 100 épocas y en las épocas 15, 25, 35 y 45, la tasa de aprendizaje se multiplica por un factor de 0,3, al alcanzarla época 55 la tasa de aprendizaje se asigna a $4 * 10^{-4}$. Se guarda la versión del modelo con la mejor precisión alcanzada.

3.3.2. Conjunto de datos

El conjunto de datos utilizado para entrenamiento es TuSimple[40], que consta de 6.408 imágenes etiquetadas de carreteras tomadas en autopistas estadounidenses en diferentes condiciones climáticas. La resolución de las imágenes es 1280×720. El conjunto de datos se compone de 3.626 para entrenamiento y 2.782 para prueba. En la figura 3.23 se puede apreciar una imagen perteneciente a TuSimple junto a su etiqueta y otras 3 variaciones generadas a partir de solo su etiqueta. Se utiliza este conjunto de datos en lugar de una basado en RUPU ya que al momento de la realización de esta memoria no existe un *dataset* basado en la superficie de RUPU, y no se crea uno debido a que hasta el momento no se ha definido como será la nueva versión de RUPU, junto con la gran cantidad de tiempo que conllevaría la creación de tal conjunto de datos. TuSimple es un conjunto de datos comúnmente utilizado en publicaciones científicas[13, 15, 20, 60, 61] y representa un escenario de mayor complejidad en comparación a la superficie de RUPU, por lo que se espera que si el modelo logra resultados satisfactorios en este, pueda replicarlos en la superficie de RUPU. Para el entrenamiento del modelo se utiliza solo el conjunto de entrenamiento compuesto por 3.626 imágenes y se reduce la resolución de las imágenes de entrada a 288 × 800. Para las etiquetas de las imágenes se crea una grilla con 56 anclas de fila (h) y 100 celdas de cuadrícula (w) para representar el área que contiene carriles. El número de carriles (c) se establece en 4.

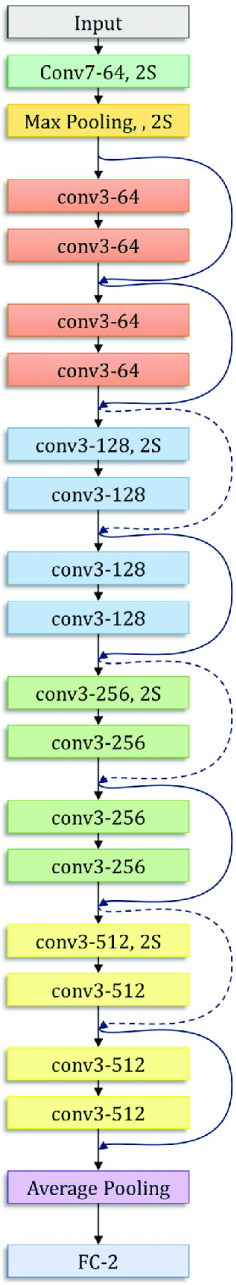


Figura 3.19: Arquitectura de Resnet 18. Fuente: [18].

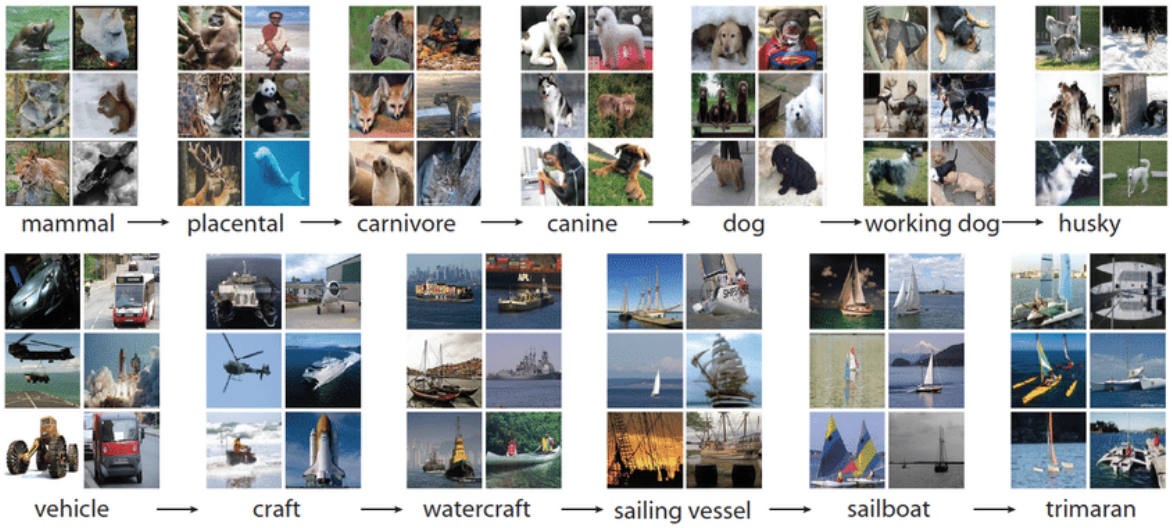


Figura 3.20: Ejemplo de imágenes pertenecientes a los árboles de animales y vehículos del dataset ImageNet. Fuente: [19].

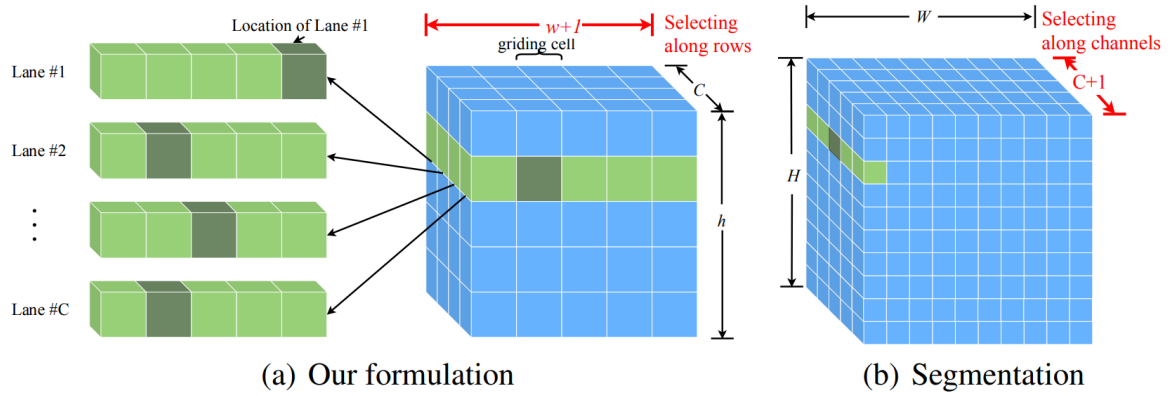


Figura 3.21: Comparación entre método de clasificación por filas y segmentación. a) Método de clasificación por filas. b) Método por segmentación. Fuente: [20].

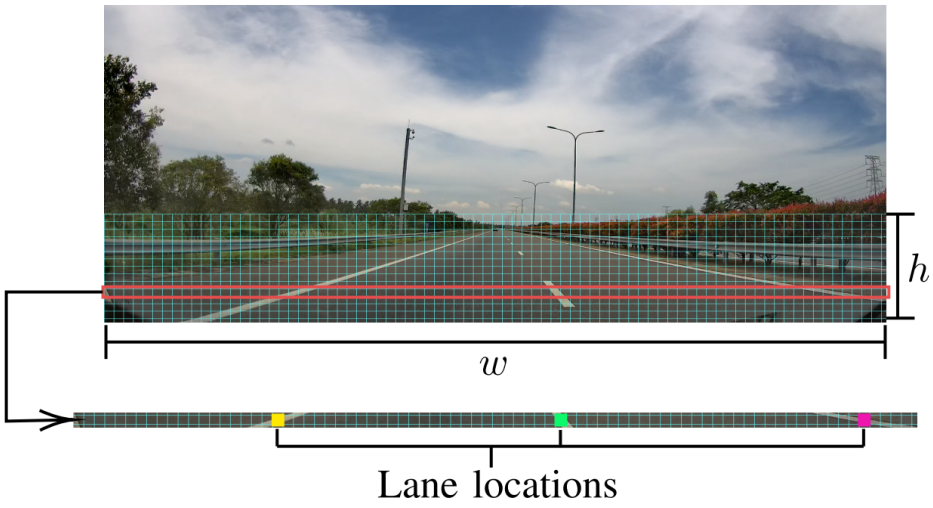


Figura 3.22: Cuadricula de salida para cada línea límite de carril en Swiftlane. Fuente: [17].

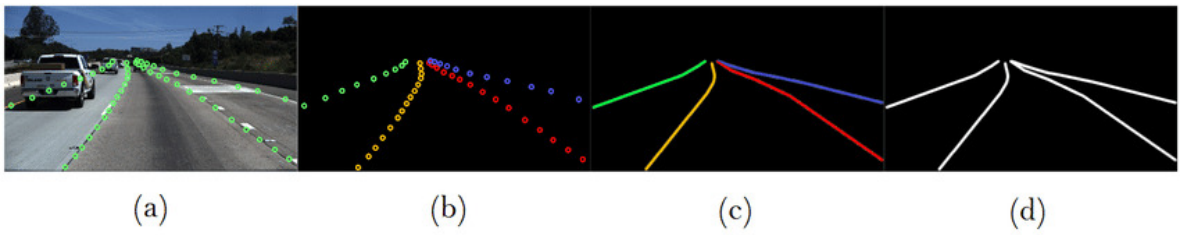


Figura 3.23: Ejemplo de imagen perteneciente a TuSimple y otras imágenes derivadas a partir de las etiquetas. Fuente: [21].

4. Experimentos y resultados

Este capítulo entrega una descripción de los procedimientos llevados a cabo para evaluar el desempeño de los métodos descritos en el capítulo anterior. También se presentan los resultados obtenidos junto con un análisis de estos. Los detalles de implementación se encuentran en repositorios públicos alojados en Github, para facilitar trabajos futuros que construyan sobre lo hecho en esta memoria.

4.1. Exactitud del detector de carriles basado en *deep learning* en *dataset* TuSimple

El detector de carriles basado en *deep learning* fue implementado en base al modelo Siwftlane y entrenado en el *dataset* TuSimple[40], como se mencionó en la sección 3. Para comprobar que el modelo logro aprender acerca de las etiquetas provistas en el *dataset*, se mide su exactitud al iniciar y finalizar su entrenamiento, es decir, en la primera época y en la época número 100. TuSimple provee la ecuación 4.11 para calcular de manera estándar la precisión del modelo.

4.1.1. Conjunto de datos

Se utiliza el conjunto de datos TuSimple, descrito en la sección 3.3.2. Se mide la precisión en el conjunto de entrenamiento y el conjunto de pruebas, compuestos por 3.626 y 2.782 imágenes respectivamente.

4.1.2. Métrica de evaluación

La métrica utilizada es la métrica oficial de rendimiento provista por los creadores del *dataset* TuSimple, la exactitud calculada con la ecuación 4.11.

$$Exactitud = \frac{\sum_{clip} C_{clip}}{\sum_{clip} S_{clip}} \tag{4.11}$$

Tabla 4.1: Exactitud en TuSimple.

	Conjunto de datos	
Época	*Entrenamiento*	*Pruebas*
1	0.481	0.4788
100	$0,988$	$0,632$

Donde C_{clip} es el número de puntos de carril pronosticados correctamente y S_{clip} es el número total de puntos reales en cada clip.

4.1.3. Resultados

Los resultados de exactitud del detector de carriles basado en *deep learning* obtenidos en la primera época y en la época número 100 del entrenamiento con el *dataset* TuSimple se muestran en la tabla 4.1.

4.2. Visualización de la salida de ambos detectores

Como se mencionó en capitulo 1, evidencia[41] reciente muestra que las métricas comúnmente utilizadas para medir el rendimiento de detectores de carriles no son aptas para tareas orientadas a conducción de vehículos. Además, las diferencias entre las salidas de ambos algoritmos complican la comparación de la precisión de estos en base a una misma métrica cuantitativa. Por lo que en esta memoria se crean visualizaciones de la salida de ambos algoritmos a partir de vídeos de entrada grabados en distintos escenarios, para así poder comparar la salida de los algoritmos cualitativamente. El detector basado en *deep learning* esta entrenado para detectar 4 líneas máximo, mientras que el detector tradicional está configurado para 3 rangos de colores distintos: cercanos al blanco, rojo y amarillo. Se mantienen estas mismas configuraciones de los detectores para todas las salidas, excepto cuando se menciona explícitamente lo contrario.

4.2.1. Conjunto de datos

Los datos de entrada a los algoritmos son vídeos grabados en diferentes escenarios. Los escenarios en los cuales se graban los vídeos de entrada apuntan a representar diferentes contextos de interés para evaluar la utilidad de los detectores de carril para RUPU. Los escenarios incluyen vídeos en contextos reales como aquellos presentes en los conjuntos de TuSimple y un vídeo grabado en una carretera vacía que no pertenece al *dataset* TuSimple. Además, se incluyen vídeos en contextos controlados, grabados en la superficie provista por RUPU. A continuación, se describen escenarios en los que se graban los vídeos de entrada:

- Escenario 1: Generado a partir de un segmento del conjunto de datos de entrenamiento del *dataset* TuSimple, consistente de imágenes obtenidas en autopistas estadounidenses en diferentes condiciones climáticas, con tráfico, curvas y movimientos de cambio de carril.
- Escenario 2: Generado a partir de un segmento del conjunto de datos de pruebas del *dataset* TuSimple, consistente de imágenes obtenidas en autopistas estadounidenses en diferentes condiciones climáticas, con tráfico, curvas y movimientos de cambio de carril.
- Escenario 3: Carretera real en buenas condiciones y sin tráfico. No pertenece al *dataset* TuSimple.
- Escenario 4: Entorno controlado. Se utiliza la superficie provista por RUPU con relativamente buena iluminación. La superficie incluye dos líneas límite de carril de color blanco y carril de color negro.
- Escenario 5: Entorno controlado. Se utiliza la superficie provista por RUPU con relativamente baja iluminación. La superficie incluye dos líneas límite de carril de color blanco y carril de color negro.
- Escenario 6: Entorno controlado. Se utiliza la superficie provista por RUPU con

relativamente buena iluminación. La superficie incluye una línea blanca que delimita un camino a seguir y el resto de la superficie es de color negro.

- Escenario 7: Entorno controlado. Se utiliza la superficie provista por RUPU con relativamente baja iluminación. La superficie incluye una línea blanca que delimita un camino a seguir y el resto de la superficie es de color negro.

Los vídeos de entrada para cada escenario se encuentran en línea [62] .

4.2.2. Resultados

Los vídeos de salida con la visualización lado a lado de ambos algoritmos se encuentran disponibles en línea [63]. La visualización de la salida del detector basado en *deep learning* se encuentra en la mitad superior de las imágenes, marcando con los colores rojo, cían, verde y amarillo los puntos pertenecientes a las 4 líneas delimitadoras de carriles de izquierda a derecha, y en caso de que para una determinada ancla de una línea delimitadora de carril no se haya detectado un punto presente en la imagen, este se marcara en el píxel extremo derecho de la imagen. La visualización de la salida del detector tradicional se encuentra en la mitad inferior de las imágenes y se marcan pequeñas rectas con colores gris claro, rojo y amarillo los segmentos de rectas pertenecientes a un rango cercano a estos colores, detectados en la imagen.

A continuación, se realizan comentarios acerca del rendimiento de ambos algoritmos en los diferentes escenarios en base algunas de las imágenes extraídas de los vídeos de salida:

- Escenario 1: El detector basado en *deep learning* logra en general un buen rendimiento en este escenario, lo cual es esperable ya que su modelo fue entrenado con este conjunto de datos y obtuvo un 0,988 de precisión. Este detector demuestra ser capaz de detectar hasta 4 líneas delimitadoras de carriles en caminos rectos y en curvas como se puede ver en las figuras 1.24 y 1.25 respectivamente. Además, es capaz de detectar las líneas delimitadoras incluso cuando estas se encuentran

cubiertas por vehículos como en las figuras 1.26 y 1.27. Pero, este detector falla cuando se realizan maniobras de cambio de línea como en las figuras 1.28, 1.29 y 1.30. El detector tradicional detecta la línea de transición entre el cielo y el piso durante la mayor parte del vídeo, como se puede ver en las figuras 1.28, 1.29, 1.30, 1.25, 1.26 y 1.27, por lo que este detector se beneficiaría del uso de una región de interés bajo el horizonte para eliminar estas detecciones.

En general el detector tradicional tiene dificultades identificando solo líneas delimitadoras de carril, esto debido a las múltiples imperfecciones con cambios de colores en el carril como en 1.24, 1.29 , 1.30 y 1.26, cambios de colores en el paisaje exterior al carril como en 1.28, 1.27, y vehículos presentes en el carril como en 1.28, 1.26 y 1.27. La gran mayoría de los segmentos de línea detectados por el detector tradicional forman parte del rango de color cercano al gris claro, y esto haría difícil distinguir cual segmento pertenece a una línea delimitadora de carril en una determinada posición, es decir, se necesitaría de operaciones posteriores sobre estos datos de salida para distinguir que puntos conforman una línea delimitadora en una determinada posición en la imagen.

- Escenario 2: Si bien el detector basado en *deep learning* obtuvo una precisión considerablemente peor en el conjunto de datos utilizado para este escenario(precisión de 0,632) en comparación al conjunto utilizado en el escenario 1(precisión de 0,988), esta diferencia entre los valores de precisión obtenidos en 4.1, no es evidente al comparar las visualizaciones de salida de ambos escenarios. Este detector logra nuevamente buenos resultados en general, detectando las líneas delimitadoras de carril en rectas como en 1.38 y 1.37, en curvas como en 1.36 e incluso cuando las líneas sean encuentran cubiertas por vehículos como en 1.35 y 1.34. Pero nuevamente falla durante la realización de maniobras de cambio de línea como en las figuras 1.31 y 1.32. En algunas ocasiones el detector falla en curvas pronunciadas y detecta una pequeña cantidad de puntos cercanos al horizonte en posiciones claramente erróneas como en la figura 1.33.

El detector tradicional sufre de las mismas falencias descritas en el escenario 1, detectando la línea de transición entre el cielo y el piso como en las figuras 1.31, 1.32, 1.33, 1.34, 1.35, 1.36, 1.37 y 1.38.Nuevamente detecta líneas que no corresponden a líneas delimitadoras de carril debido a cambios de colores en el paisaje exterior al carril como en 1.31, 1.32, 1.33, 1.34, 1.35, 1.36, 1.37 y 1.38, y vehículos presentes en el carril como en 1.35, 1.32 y 1.31.

- Escenario 3: El detector basado en *deep learning* no es capaz de detectar las líneas delimitadoras de carril en este escenario, como se puede ver en la figura 1.39.
  El detector tradicional logra detectar las líneas delimitadoras de carril correctamente, haciendo uso los rangos de colores cercanos al amarillo y gris claro cómo se puede ver en la figura 1.39. Si bien este escenario está basado en un ambiente real, similar a los escenarios 1 y 2, los resultados de los detectores cambiaron considerablemente. El detector tradicional logra un mejor rendimiento debido a las buenas condiciones de la carretera y la escasez de vehículos en ella, mientras que en el caso del detector basado en *deep learning*, su falla podría ser explicada debido a que los filtros de sus capas convolucionales entrenados en el *dataset* TuSimple no son aptos para este escenario.

- Escenario 4: El detector basado en *deep learning* no logra detectar las líneas delimitadoras de carril, como se ve en las figuras 1.41, 1.40 y 1.42.
  El detector tradicional logra detectar las líneas delimitadoras de carril correctamente, como se puede ver en 1.42. Un problema para el detector tradicional es la presencia de un obstáculo como otro robot móvil en el carril, ya que el detector no puede detectar el carril cubierto por el obstáculo, como en la figura 1.41. Este problema puede disminuirse en RUPU al utilizar robots móviles que contenga huecos que permitan visualizar las líneas delimitadoras de carril aun cuando estos la cubran, como en 1.40. También es importante recalcar que objetos que contengan colores pertenecientes al rango de colores de detección configurados en el detector tradicional serán detectados por este, como los cables rojos del robot

móvil en 1.40.

- Escenario 5: El detector basado en *deep learning* no logra detectar las líneas delimitadoras de carril, como se ve en la figura 1.43.
  En este escenario la iluminación de la superficie disminuye considerablemente en comparación al escenario 4 y el detector tradicional ya no es capaz de detectar las líneas delimitadoras de carril, requiriendo ajustes a los valores límites de sus rangos de colores en HSV.

- Escenario 6: El detector basado en *deep learning* no logra detectar las líneas delimitadoras de carril, como se ve en las figuras 1.45 y 1.46.
  El detector tradicional logra resultados similares a los vistos en el escenario 4, siendo capaz de detectar la línea en 1.46 y siendo incapaz de detectar las partes de la línea que se encuentran obstaculizadas por objetos como en 1.45. La presencia de objetos fuera del carril con colores dentro del rango de colores configurado inicialmente en el detector tradicional, producen falsas detecciones en 1.45, 1.46, y deberían ser minimizados en el ambiente controlado de RUPU si se desea utilizar este detector.

- Escenario 7: El detector basado en *deep learning* no logra detectar las líneas delimitadoras de carril, como se ve en las figuras 1.47.
  Similarmente a lo ocurrido en el escenario 5, al disminuir la iluminación de la superficie considerablemente en comparación al escenario 6, el detector tradicional ya no es capaz de detectar las líneas delimitadoras de carril, requiriendo ajustes a los valores límites de sus rangos de colores en HSV.

El detector basado en *deep learning* demostró ser lo suficientemente robusto como para detectar líneas delimitadoras de carril, aun cuando la visión de estas se encuentra obstruida, en los dos escenarios reales para los cuales fue entrenado (escenarios 1 y 2 basados en TuSimple). Sin embargo, los filtros de la red neuronal entrenados solo en un conjunto de datos de relativamente mayor complejidad no son lo suficientemente ge-

nerales como para que el algoritmo detecte líneas delimitadoras de carril en escenarios relativamente más simples (escenarios 3, 4, 5, 6 y 7). Esto sugiere que el detector basado en *deep learning* requiere ser entrenado en un conjunto de datos basado en el escenario que el detector será usado, aun cuando este ya ha sido entrenado y ha mostrado funcionar correctamente en escenarios de mayor complejidad. Para mejorar el rendimiento de la red neuronal en la superficie de RUPU se podría reentrenar la red con un conjunto de datos basado en RUPU, pero hasta el momento tal *dataset* no existe, y crearlo conlleva una cantidad de tiempo que escapa a la duración considerada de esta memoria. Basado en lo anterior y en los malos resultados que este detector obtuvo en los escenarios de 4 a 7, se decide que no es apto para la plataforma RUPU en su iteración actual y no se implementa en la tarjeta Jetson TX2 para realizar pruebas del tiempo de inferencia.

Si bien el detector tradicional no obtuvo buenos resultados en los escenarios de mayor complejidad (1 y 2), esto no representa un problema mayor para su implementación en el ambiente controlado de RUPU, como se evidencia en las imágenes de resultados en los escenarios 4 y 6. El detector tradicional detector posee algunas dificultades para percibir las líneas delimitadoras de carril en determinadas condiciones y no es trivial determinar cuáles de los puntos detectados pertenecen a una determinada línea delimitadora, pero estas dificultades pueden ser superadas al utilizar un ambiente controlado como RUPU en conjunto con una configuración del detector específica para tal ambiente. Por ejemplo, es posible demarcar las líneas delimitadoras de carriles en la superficie con diferentes colores para así poder distinguir con mayor facilidad cuales de los puntos detectados pertenecen a una determinada línea, y utilizar robots móviles con colores diferentes a las líneas para que estos no entreguen falsas detecciones, además se puede disminuir la cantidad de objetos externos al carril con colores similares a las líneas delimitadoras. Un ejemplo de adaptar la configuración del detector tradicional a un determinado ambiente para mejorar sus resultados se puede ver en la figura 1.44, donde se modifica la configuración del detector para que este sea capaz de detectar solo rangos de colores cercanos al blanco en la luminosidad disminuida del escenario 5 con

relación al escenario 4. El detector tradicional se implementa en la tarjeta Jetson TX2 para realizar pruebas de latencia.

4.3. Latencia

La latencia o tiempo de inferencia es uno de los aspectos más cruciales de la implementación de los algoritmos en un entorno de conducción autónoma, en especial en el caso de ser utilizados en sistemas embebidos. Se mide la latencia de ambos detectores en diferentes escenarios, en el mismo computador personal con procesador AMD Ryzen 9 5980HS y GPU NVIDIA GEFORCE GTX 1650, para registrar si estos afectan los resultados. También se miden los resultados del detector tradicional implementado en la tarjeta Jetson TX2, con dos configuraciones distintas, una para detectar 3 colores diferentes(blanco, rojo y amarillo) y otra para detectar solo uno(blanco). El detector tradicional está configurado para hacer uso de la unidad de procesamiento central(CPU), mientras que el detector basado *deep learning* también hace uso de unidades de cómputo paralelo(GPU).

4.3.1. Conjunto de datos

Se utiliza como entrada a los algoritmos 100 imágenes extraídas de vídeos en distintos escenarios. Los escenarios se describen a continuación:

- Escenario 1: Generado a partir de un segmento del conjunto de datos de entrenamiento del *dataset* TuSimple, consistente de imágenes obtenidas en autopistas estadounidenses en diferentes condiciones climáticas, con tráfico, curvas y movimientos de cambio de carril.
- Escenario 2: Generado a partir de un segmento del conjunto de datos de pruebas del *dataset* TuSimple, consistente de imágenes obtenidas en autopistas estadounidenses en diferentes condiciones climáticas, con tráfico, curvas y movimientos de cambio de carril.
- Escenario 3: Carretera real en buenas condiciones y sin tráfico. No pertenece al *dataset* TuSimple.
- Escenario 4: Entorno controlado. Se utiliza la superficie provista por RUPU con

relativamente buena iluminación. La superficie incluye dos líneas límite de carril de color blanco y carril de color negro.

- Escenario 5: Entorno controlado. Se utiliza la superficie provista por RUPU con relativamente buena iluminación. La superficie incluye una línea blanca que delimita un camino a seguir y el resto de la superficie es de color negro.

Los vídeos se encuentran disponibles en línea[64].

4.3.2. Métrica

La latencia de ambos algoritmos se mide realizando 100 ejecuciones donde los algoritmos detectores de carriles procesan distintas imágenes de entrada, guardando como latencia el mayor tiempo de ejecución obtenido, mientras que la latencia promedio se calcula al promediar los tiempos medidos para las 100 ejecuciones. Al realizar las mediciones se utiliza una ejecución como calentamiento para el hardware, la cual no es incluida dentro de las mediciones, ya que los dispositivos modernos pueden tener múltiples estados de consumo de poder reduciendo su capacidad de cómputo cuando no están siendo utilizados, lo cual produce que generalmente la primera ejecución sea mucho más lenta en comparación al resto.

Los tiempos de ejecución son medidos utilizando el paquete "time" de python. Para el detector basado en *deep learning* se mide el tiempo de ejecución de la inferencia del modelo:

```
start = time.time()
model(input_tensor)
end = time.time()
```

En el caso del detector tradicional, el tiempo que este tarda en procesar una imagen depende de la cantidad de diferentes colores que hayan sido configurados para ser detectados, por lo que se mide el tiempo de ejecución con el siguiente código:

```
start = time.time()
detector.setImage(frame)
```

```
detections = {
    color: detector.detectLines(ranges) for color, ranges in list(color_ranges.items()
    )
}
end = time.time()
```

4.3.3. Resultados

La latencia o valor máximo $max(t)$ de los tiempos de ejecución t, junto con la latencia promedio $\bar{t}$ medidas en el computador de escritorio y en la tarjeta Jetson TX2 se representan en las tablas 4.2 y 4.3 respectivamente.

En la tabla 4.2 se observa que las latencias de ambos detectores se encuentran en el mismo orden de magnitud, siendo 4,034[ms] y 8,002[ms] los valores más altos obtenidos del detector basado en *deep learning* y el detector tradicional respectivamente. Los valores obtenidos muestran que actualmente es posible obtener latencias de detectores basados *deep learning* competitivas con las de detectores tradicionales, al hacer uso de unidades de procesamiento paralelo y arquitecturas de redes neuronales relativamente compactas.

Tanto en la tabla 4.2 como en la tabla 4.3 se puede observar que la latencia del detector tradicional se ve afectada por la cantidad de colores configurados a detectar y el escenario en el que se está utilizando, siendo estos efectos aún más evidentes en la tarjeta Jetson TX2. La latencia promedio puede ser disminuida al configurar menos colores a detectar y al utilizar el detector en escenarios controlados en los que se minimiza las variaciones de colores.

Tabla 4.2: Latencia de detectores de carril en computador de escritorio.

	Tradicional (3 Colores)		Tradicional (1 Color)		Deep learning	
Escenario	$\bar{t}[ms]$	$max(t)[ms]$	$\bar{t}[ms]$	$max(t)[ms]$	$\bar{t}[ms]$	$max(t)[ms]$
1	$6,340$	$8,002$	$4,410$	$5,002$	$3,400$	$4,016$
2	$6,520$	$8,000$	$4,630$	$6,000$	$3,453$	$4,034$
3	$5,780$	$7,000$	$3,580$	$4,003$	$3,494$	$4,026$
4	$5,681$	$7,004$	$3,800$	$5,000$	$3,290$	$4,022$
5	$5,720$	$8,000$	$3,490$	$4,004$	$3,391$	$4,019$

Tabla 4.3: Latencia de detectores de carril en Jetson TX2.

	Tradicional (3 Colores)		Tradicional (1 Color)	
Escenario	$\bar{t}[ms]$	$max(t)[ms]$	$\bar{t}[ms]$	$max(t)[ms]$
1	$32,424$	$47,216$	$28,444$	$43,824$
2	$36,328$	$43,891$	$32,049$	$38,02$
3	$27,241$	$31,430$	$17,525$	$18,738$
4	$26,146$	$31,637$	$19,921$	$22,081$
5	$23,537$	$29,062$	$17,503$	$25,201$

5. Conclusiones y trabajo futuro

La motivación de esta memoria era el estudio, implementación y evaluación de algoritmos de detección de carriles para ser utilizados en la plataforma RUPU. Para este fin, se realizó un estudio previo de diversas alternativas de diseño de detectores de carriles basados en cámaras tradicionales y visión por computador. Del estudio se decidió implementar dos detectores de carriles, uno tradicional y otro basado en deep learning, para posteriormente evaluar si estos son adecuados para la plataforma RUPU. Los puntos de interés a evaluar son la latencia y qué tan capaces son de detectar los carriles presentes en el escenario que se encuentran. La capacidad de los algoritmos para detectar carriles en diferentes escenarios fue evaluada cualitativamente mediante visualizaciones de las salidas de los algoritmos en diferentes escenarios, debido a que evidencia [41] reciente muestra que las métricas comúnmente utilizadas para medir el rendimiento de detectores de carriles no son aptas para tareas orientadas a la conducción de vehículos. Similarmente, la latencia se mide en distintos escenarios y configuraciones para observar cómo afectan los resultados obtenidos. Los datos obtenidos otorgan una perspectiva acerca de las ventajas y desventajas de cada algoritmo y permiten concluir cuál de ellos es preferible implementar en RUPU.

En base a los datos obtenidos en esta memoria, se recomienda implementar el detector tradicional en RUPU, desarrollando la nueva versión de RUPU en conjunto con el detector para mitigar las falencias de este último. Las recomendaciones para desarrollar ambos en conjunto son las siguientes:

- Utilizar una zona de interés (ROI) para disminuir falsos positivos, por ejemplo, eliminando la parte superior al horizonte en la imagen de entrada.
- Utilizar líneas delimitadoras de carril de distintos colores en la superficie de RUPU en conjunto con un relleno de carril de diferente color para identificar sus posiciones relativas a otras líneas con mayor facilidad. Por ejemplo, en una vía con dos carriles demarcados por 3 líneas, se puede utilizar el color amarillo para

marcar la línea central, mientras que las exteriores se marcan con blanco y el relleno del carril es negro. De esta manera, también se disminuye la necesidad de procesamiento posterior a la salida del detector.

- Utilizar robots móviles y obstáculos de colores diferentes a los utilizados en las líneas para que estos no entreguen falsos positivos.
- Minimizar la cantidad de colores configurados a detectar por el detector para así disminuir su latencia.
- Utilizar iluminación adecuada en el escenario donde sea posible para que el detector pueda distinguir entre los diferentes colores presentes en la imagen de entrada.
- Este detector no es capaz de detectar las partes de las líneas delimitadoras de carril cuando la visión a estas se encuentra obstruida y, por ende, se debe planificar para esta eventualidad.
- El detector es capaz de obtener valores de latencia en el orden de magnitud de 10 [ms] en una tarjeta Jetson TX2 sin utilizar unidades de procesamiento paralelo, por lo que se puede disminuir el costo de los nuevos robots móviles de RUPU al no integrar unidades de procesamiento paralelo

En cuanto a posibles trabajos futuros, una vez se haya finalizado el desarrollo de la nueva superficie de RUPU, se recomienda crear un conjunto de datos de imágenes obtenidas en la superficie, que permitiría entrenar algoritmos basados en *deep learning* para realizar futuras adiciones a RUPU. El detector basado en *deep learning* utilizado en esta memoria mostró obtener latencias comparables a las del detector tradicional en el computador de escritorio, además de ser capaz de detectar líneas delimitadoras de carril aun cuando la visión a estas se encuentra obstruida en escenarios reales. Si bien explicar el comportamiento del detector basado en *deep learning* no es trivial, este muestra gran potencial para funcionar en condiciones donde el detector tradicional falla. Además, el área de los detectores de carril basados en *deep learning* se encuentra actualmente en constante desarrollo de nuevos y mejores algoritmos, por lo que implementar uno de

estos en el futuro puede resultar beneficioso. El conjunto de datos también sería útil para implementar algoritmos de visión que funcionen en paralelo al detector y sean capaces de detectar obstáculos en el camino. También se recomienda desarrollar o utilizar una nueva métrica que se adecue a tareas orientadas a la conducción de vehículos, para así evaluar cuantitativamente la precisión de los próximos detectores de carril.

Apéndice A Visualización de salida de detectores de carril

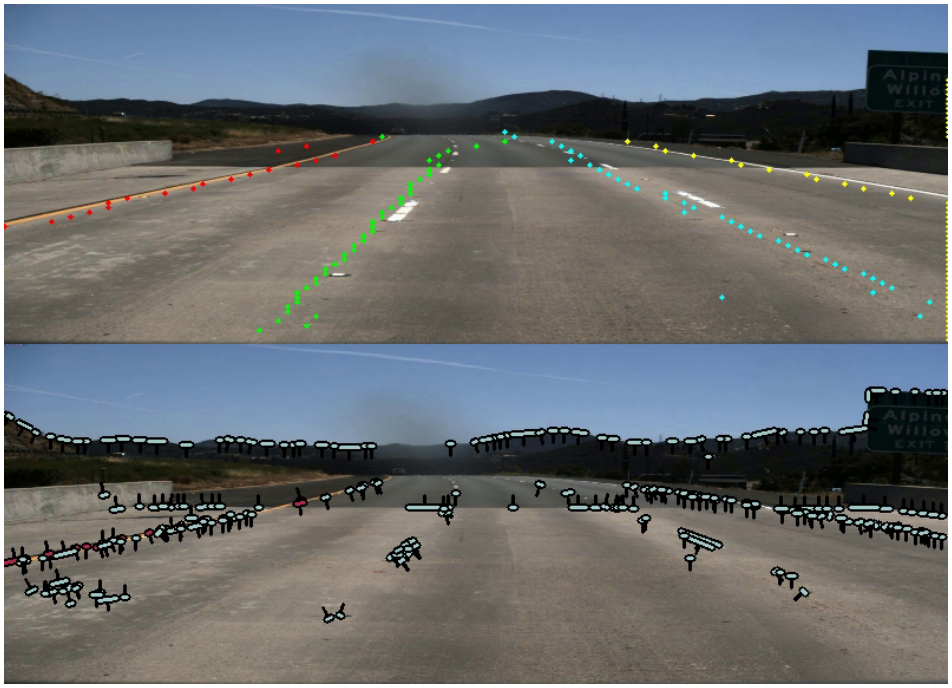


Figura 1.24: Escenario 1: Carril recto. Fuente: Elaboración propia.

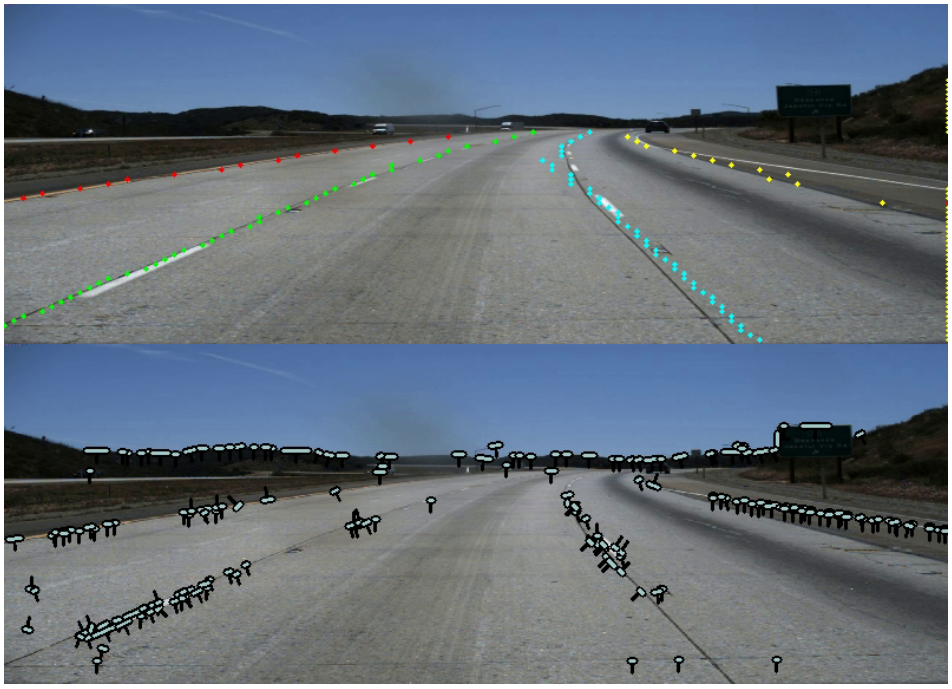


Figura 1.25: Escenario 1: Carril con curva hacia la derecha. Fuente: Elaboración propia.

Figura 1.26: Escenario 1: Carril con curva hacia la izquierda. Fuente: Elaboración propia.

Figura 1.27: Escenario 1: Carril con curva y vehículos cubriendo líneas delimitadoras de carril. Fuente: Elaboración propia.

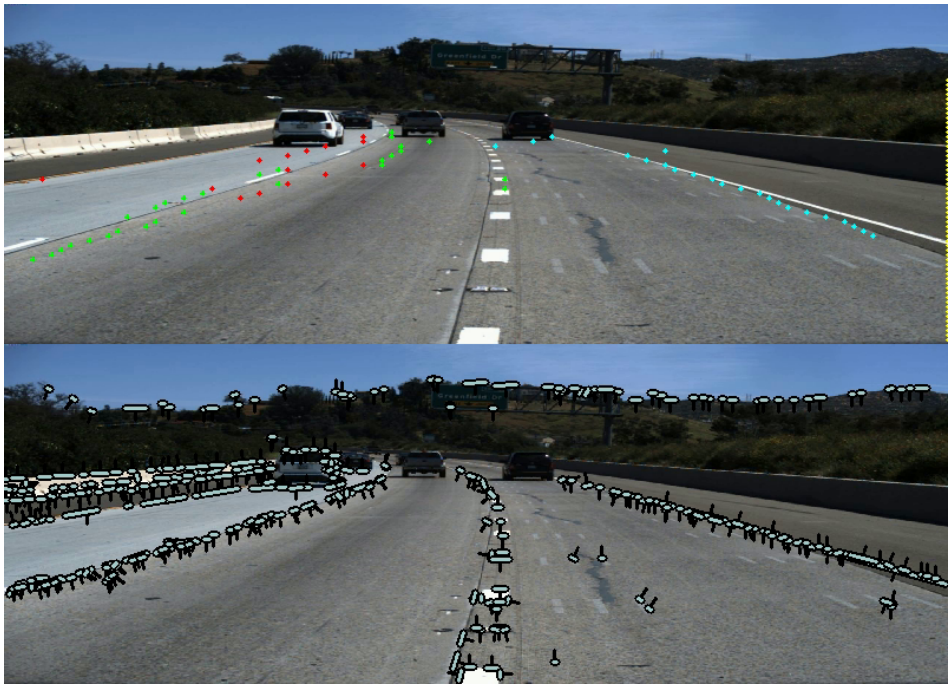


Figura 1.28: Escenario 1: Maniobra de cambio de línea 1. Fuente: Elaboración propia.

Figura 1.29: Escenario 1: Maniobra de cambio de línea 2. Fuente: Elaboración propia.

Figura 1.30: Escenario 1: Maniobra de cambio de línea 3. Fuente: Elaboración propia.

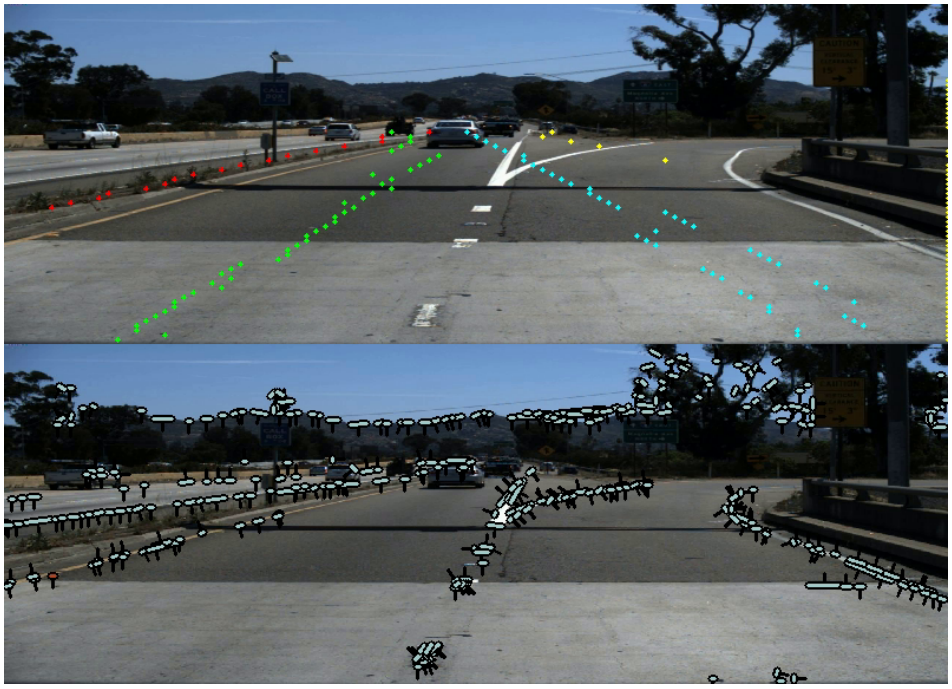


Figura 1.31: Escenario 2: Maniobra de cambio de línea 1. Fuente: Elaboración propia.

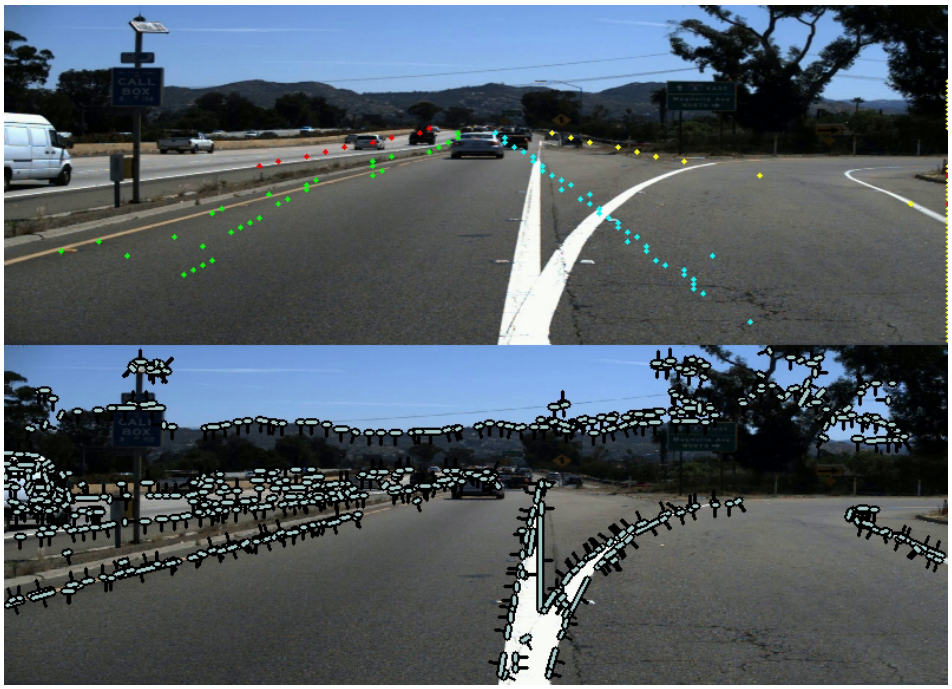


Figura 1.32: Escenario 2: Maniobra de cambio de línea. Fuente: Elaboración propia.

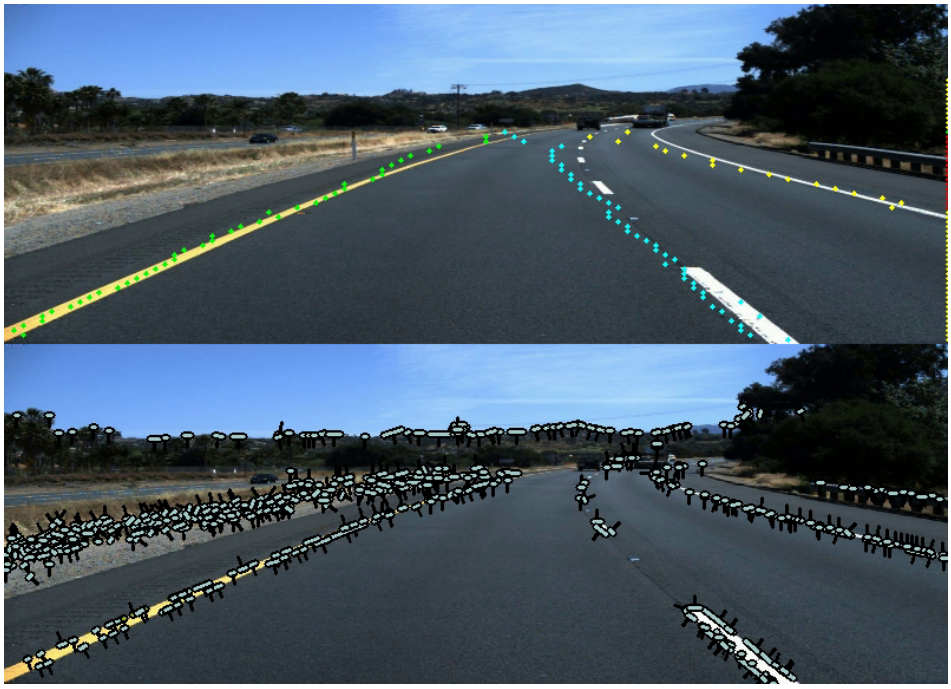


Figura 1.33: Escenario 2: Carril con curva hacia la derecha 1. Fuente: Elaboración propia.

Figura 1.34: Escenario 2: Carril recto con vehículos cubriendo líneas delimitadoras de carril. Fuente: Elaboración propia.

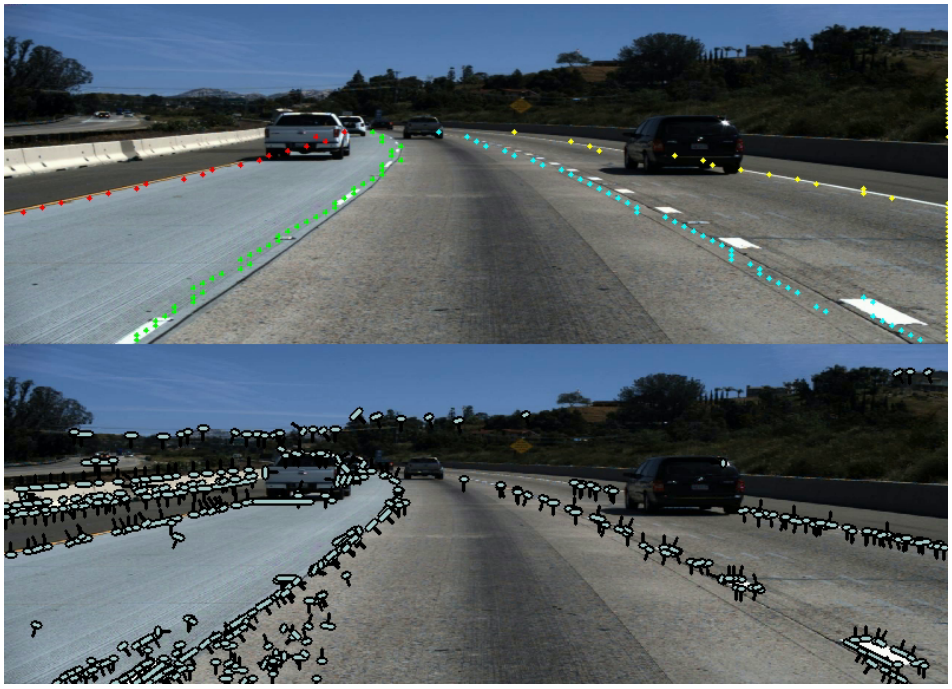


Figura 1.35: Escenario 2: Carril con curva hacia la izquierda. Fuente: Elaboración propia.

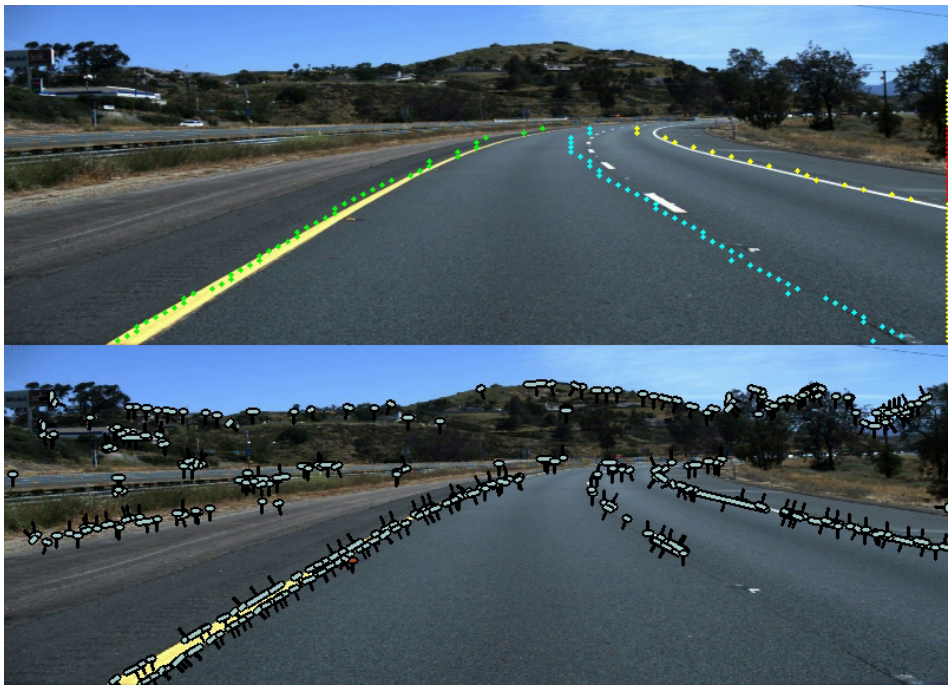


Figura 1.36: Escenario 2: Carril con curva hacia la derecha 2. Fuente: Elaboración propia.

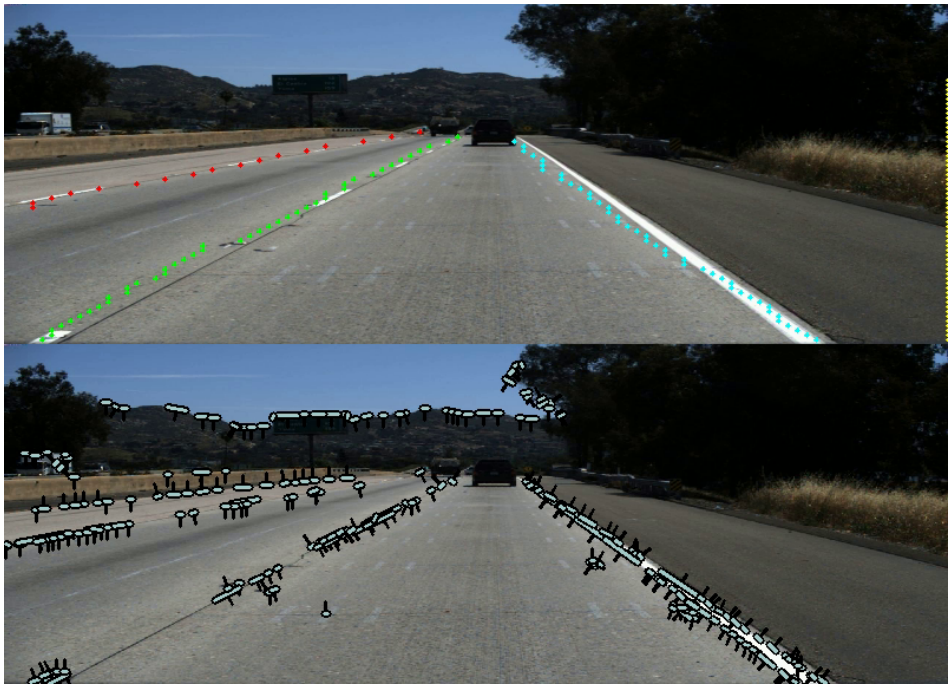


Figura 1.37: Escenario 2: Carril recto 1. Fuente: Elaboración propia.

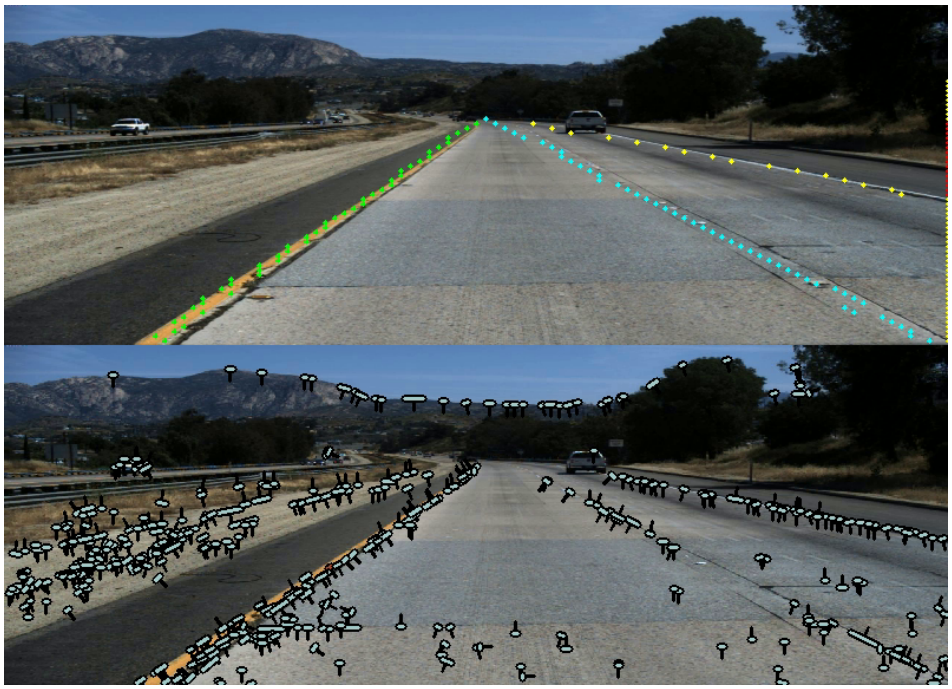


Figura 1.38: Escenario 2: Carril recto 2. Fuente: Elaboración propia.

Figura 1.39: Escenario 3: Carril recto. Fuente: Elaboración propia.

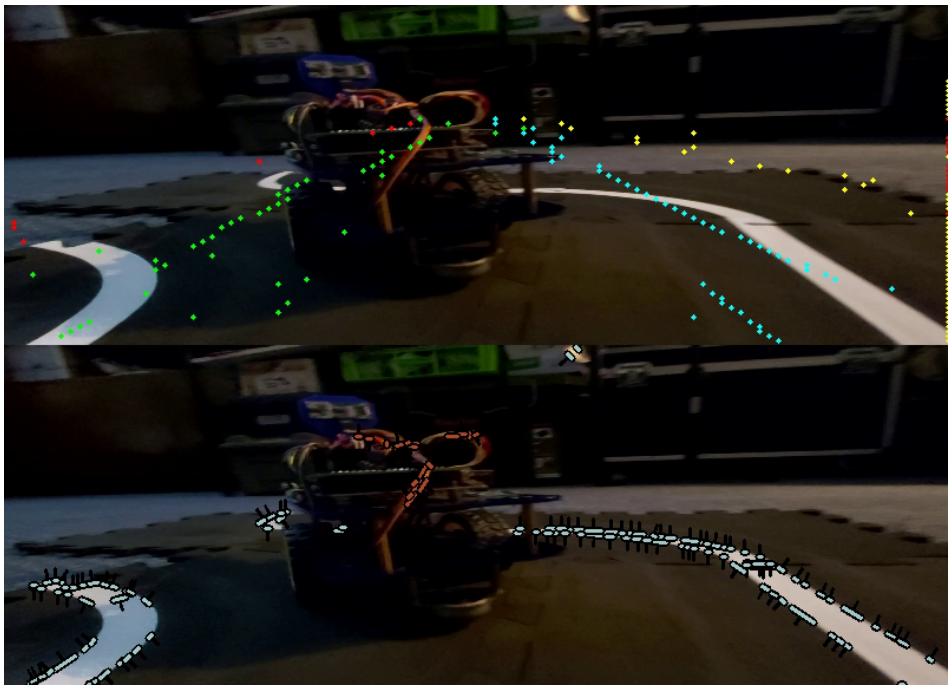


Figura 1.40: Escenario 4: Curva en superficie RUPU con robot 1. Fuente: Elaboración propia.

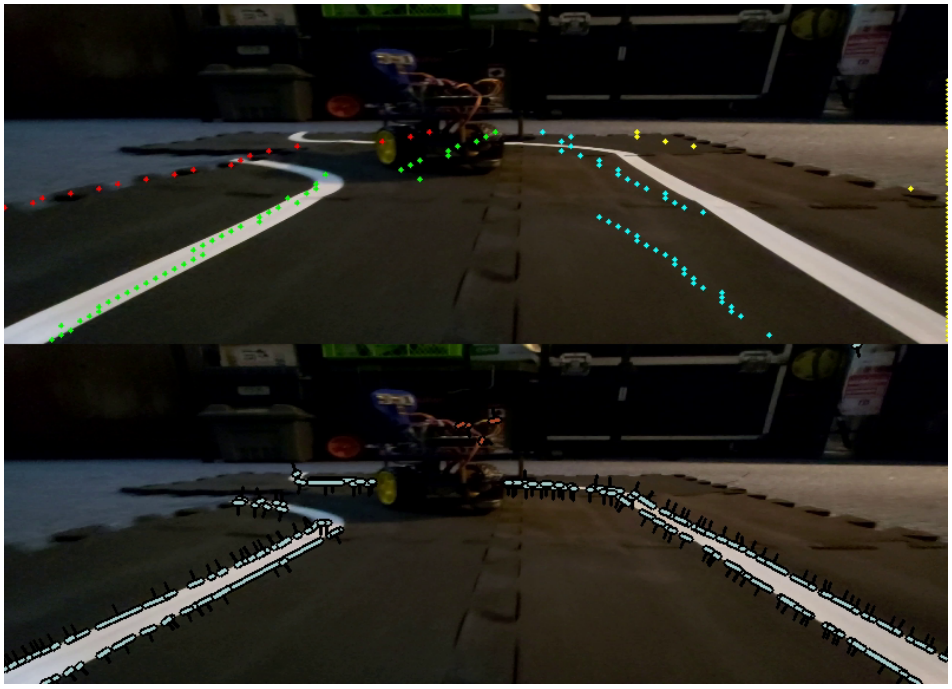


Figura 1.41: Escenario 4: Curva en superficie RUPU con robot 2. Fuente: Elaboración propia.

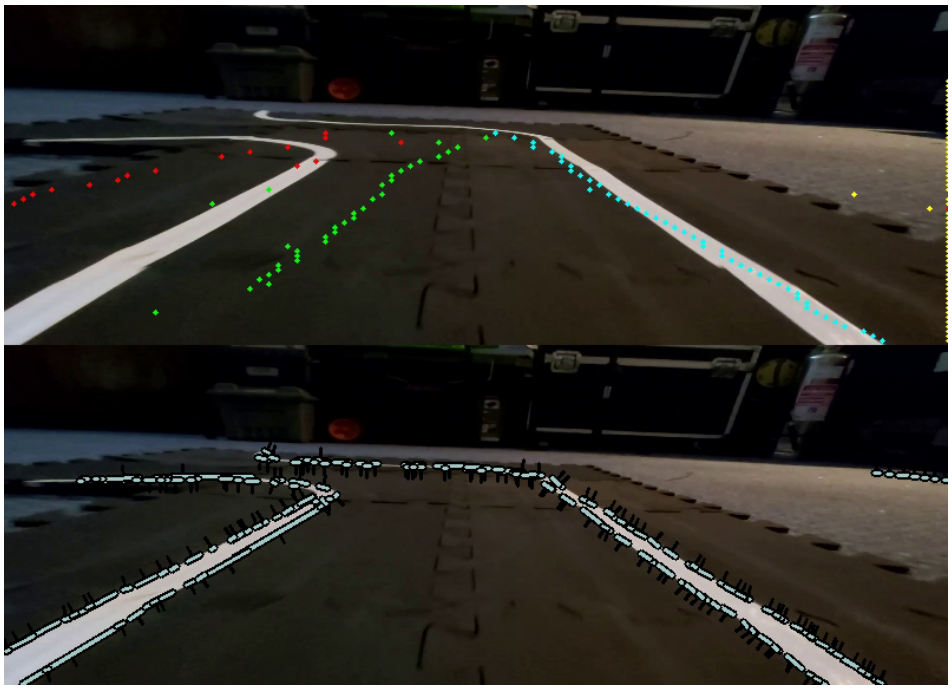


Figura 1.42: Escenario 4: Curva en superficie RUPU. Fuente: Elaboración propia.

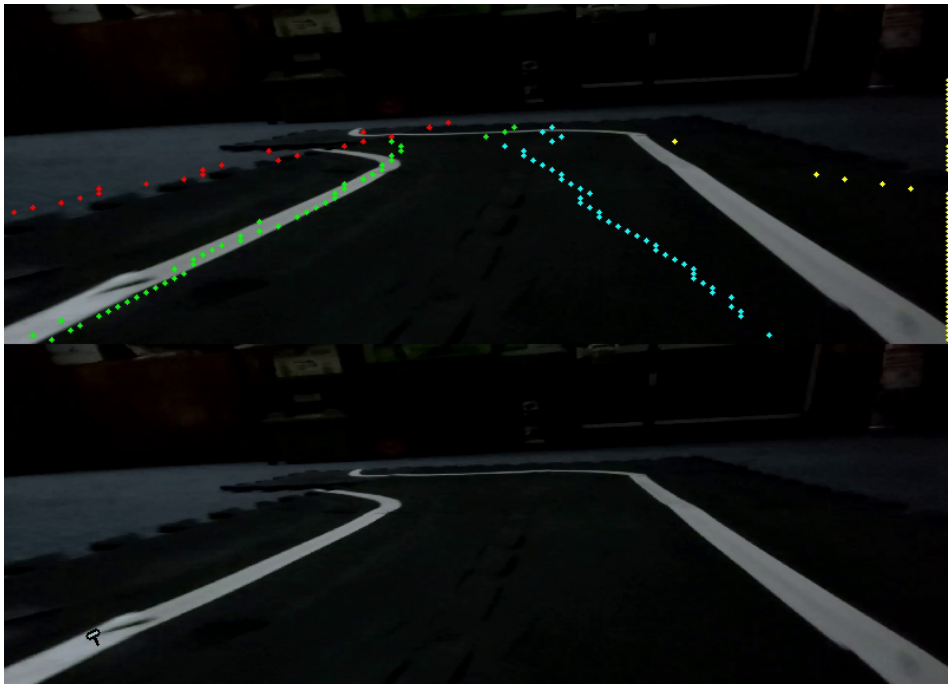


Figura 1.43: Escenario 5: Curva en superficie RUPU. Fuente: Elaboración propia.

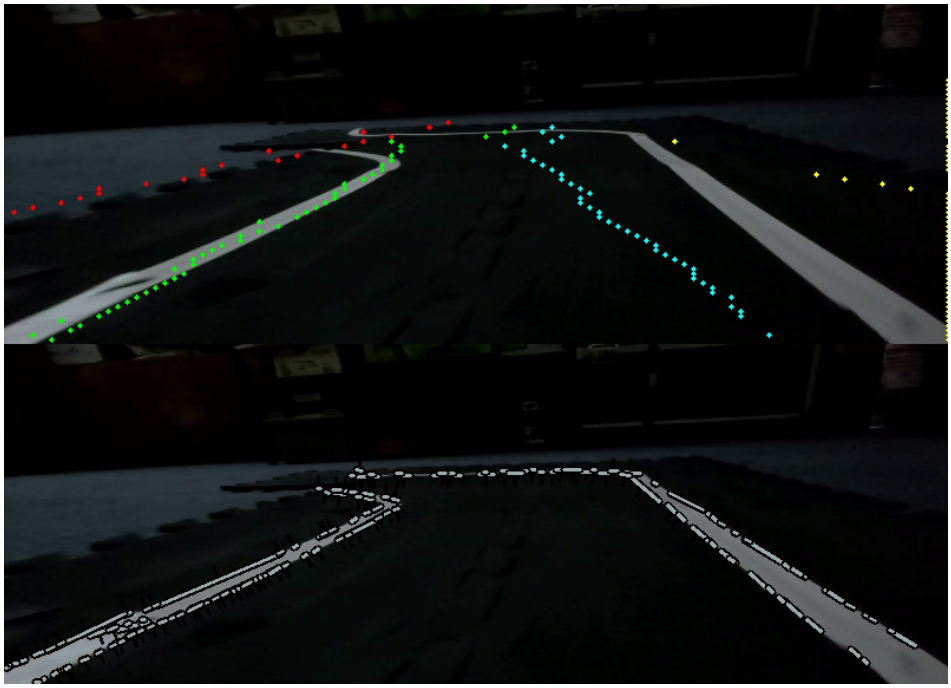


Figura 1.44: Escenario 5: Curva en superficie RUPU con nueva configuración del detector tradicional. Fuente: Elaboración propia.

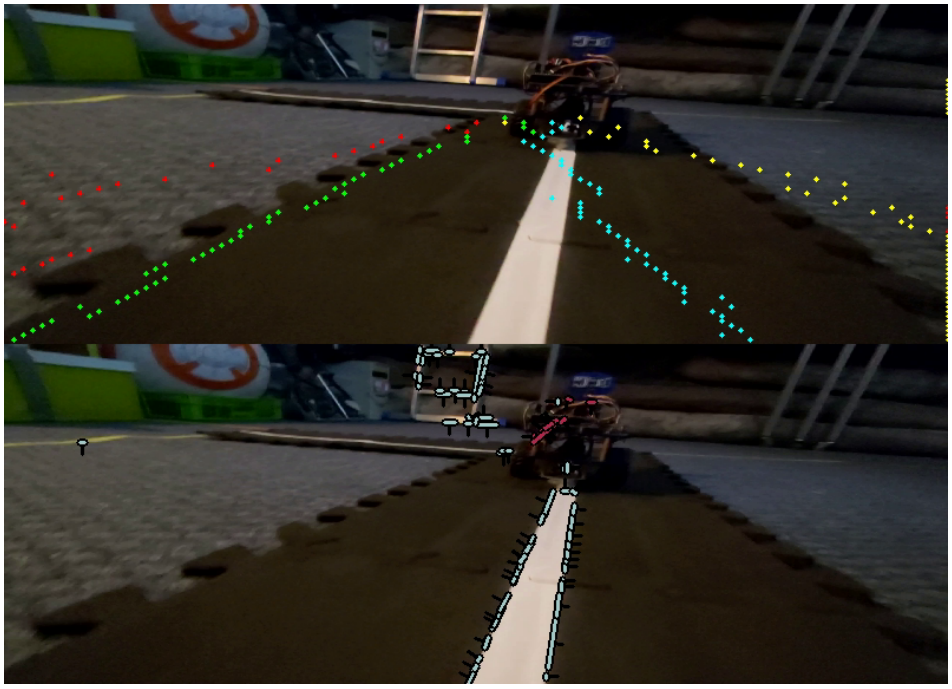


Figura 1.45: Escenario 6: Recta en superficie RUPU con robot. Fuente: Elaboración propia.

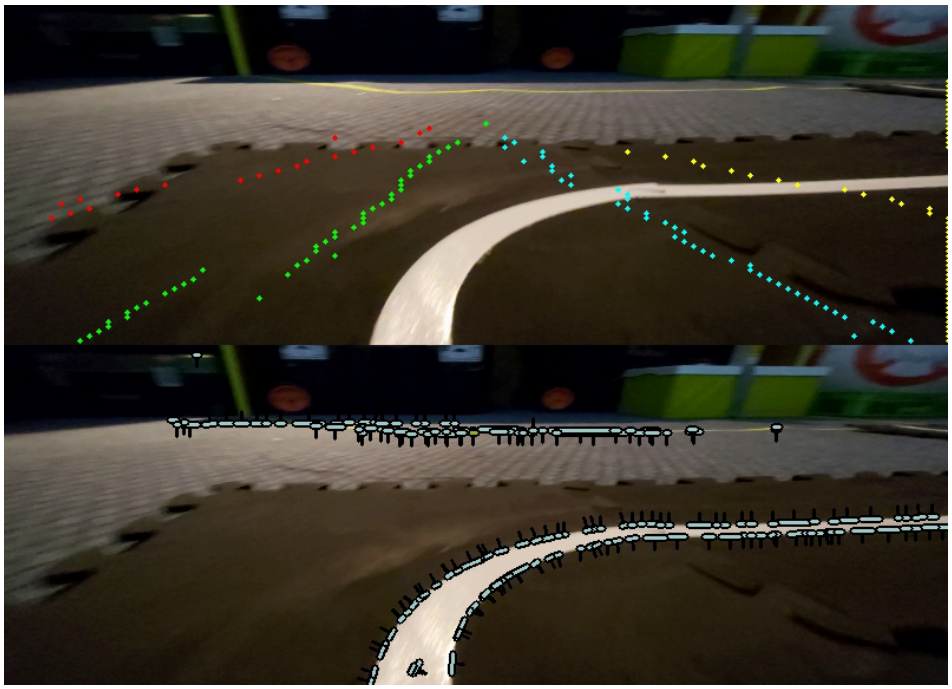


Figura 1.46: Escenario 6: Curva en superficie RUPU. Fuente: Elaboración propia.

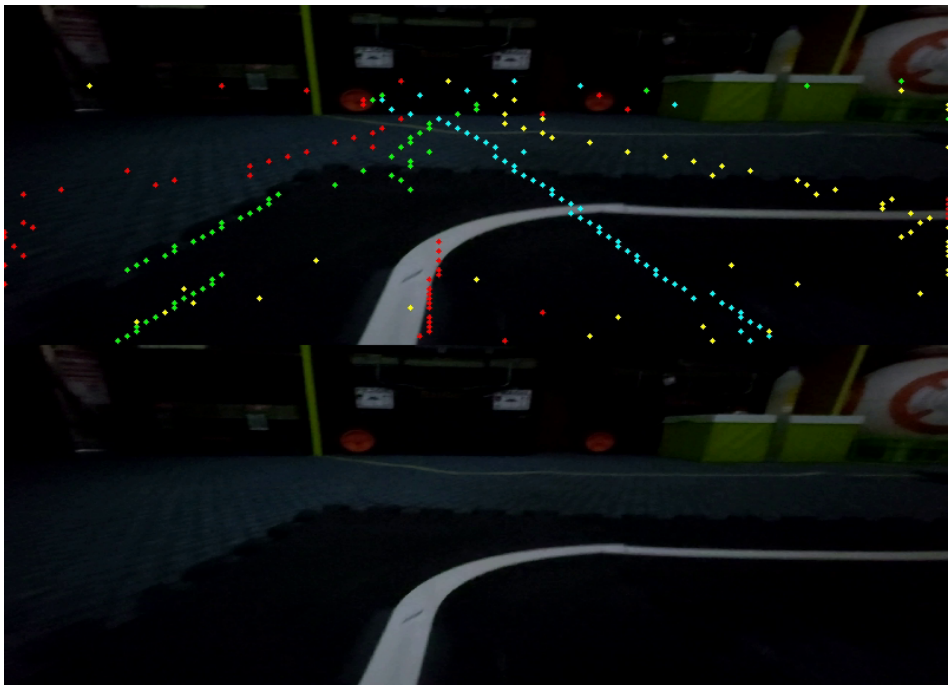


Figura 1.47: Escenario 7: Curva en superficie RUPU. Fuente: Elaboración propia.

Referencias

[1] C. Escobar, "Diseño, implementación y evaluación de una plataforma experimental para el estudio del control de una flota de robots móviles," Master's thesis, Departamento de Electrónica. Universidad Técnica Federico Santa María, Noviembre 2021.

[2] "Hsb/hsv und hsl-farbmodell," Oct 2008. [Online]. Available: https://wisotop.de/hsv-und-hsl-farbmodell.php

[3] Duckietown, "Duckietown object detection dataset," https://universe.roboflow.com/duckietown-phrqi/duckietown-object-detection, jun 2022, visited on 2022-11-28. [Online]. Available: https://universe.roboflow.com/duckietown-phrqi/duckietown-object-detection

[4] T. Kacmajor, "Hough lines transform explained," Nov 2020. [Online]. Available: https://medium.com/@tomasz.kacmajor/hough-lines-transform-explained-645feda072ab

[5] G. J. Bergues, C. Schürrer, and N. Brambilla, "Straight line detection through subpixel hough transform," in *Intelligent Computing: Proceedings of the 2019 Computing Conference, Volume 2*. Springer, 2019, pp. 1129–1137.

[6] "Convolutional neural networks - basics." [Online]. Available: https://mlnotebook.github.io/post/CNN1/

[7] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[8] E. Anello, "Visualizing the feature maps and filters by convolutional neural networks," Nov 2022. [Online]. Available: https://medium.com/dataseries/visualizing-the-feature-maps-and-filters-by-convolutional-neural-networks-e1462340518e

[9] "Redes neuronales convolucionales." [Online]. Available: https://es.mathworks.com/discovery/convolutional-neural-network-matlab.html

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[11] T. Hoeser and C. Kuenzer, "Object detection and image segmentation with deep learning on earth observation data: A review-part i: Evolution and recent trends," *Remote Sensing*, vol. 12, 05 2020.

[12] G. Liu, F. Wörgötter, and I. Markelić, "Combining statistical hough transform and particle filter for robust lane detection and tracking," in *2010 IEEE Intelligent Vehicles Symposium*. IEEE, 2010, pp. 993–997.

[13] Z. Feng, S. Guo, X. Tan, K. Xu, M. Wang, and L. Ma, "Rethinking efficient lane detection via curve modeling," *arXiv preprint arXiv:2203.02431*, 2022.

[14] L. Tabelini, R. Berriel, T. M. Paixao, C. Badue, A. F. De Souza, and T. Oliveira-Santos, "Keep your eyes on the lane: Real-time attention-guided lane detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 294–302.

[15] D. Neven, B. De Brabandere, S. Georgoulis, M. Proesmans, and L. Van Gool, "Towards end-to-end lane detection: an instance segmentation approach," in *2018 IEEE intelligent vehicles symposium (IV)*. IEEE, 2018, pp. 286–291.

[16] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. F. Chen, C. Choi, J. Dusek, Y. Fang *et al.*, "Duckietown: an open, inexpensive and flexible platform for autonomy education and research," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1497–1504.

[17] O. Jayasinghe, D. Anhettigama, S. Hemachandra, S. Kariyawasam, R. Rodrigo, and P. Jayasekara, "Swiftlane: Towards fast and efficient lane detection," in *2021*

20th IEEE International Conference on Machine Learning and Applications (ICMLA). IEEE, 2021, pp. 859–864.

[18] R. Kundu, R. Das, Z. W. Geem, G.-T. Han, and R. Sarkar, "Pneumonia detection in chest x-ray images using an ensemble of deep learning models," *Plos one*, vol. 16, no. 9, p. e0256630, 2021.

[19] T. Ye, "Visual object detection from lifelogs using visual non-lifelog data," Ph.D. dissertation, Dublin City University, 2018.

[20] Z. Qin, H. Wang, and X. Li, "Ultra fast structure-aware deep lane detection," in *European Conference on Computer Vision*. Springer, 2020, pp. 276–291.

[21] S. Chougule, N. Koznek, A. Ismail, G. Adam, V. Narayan, and M. Schulze, "Reliable multilane detection and classification by utilizing cnn as a regression network," in *proceedings of the european conference on computer vision (ECCV) workshops*, 2018, pp. 0–0.

[22] F. Duarte and C. Ratti, "The impact of autonomous vehicles on cities: A review," *Journal of Urban Technology*, vol. 25, no. 4, pp. 3–18, 2018.

[23] "Waymo driver." [Online]. Available: https://waymo.com/intl/es/waymo-driver/

[24] "Autopilot and full self-driving capability," Aug 2022. [Online]. Available: https://www.tesla.com/en_AE/support/autopilot-and-full-self-driving-capability

[25] S. Campbell, N. O'Mahony, L. Krpalcova, D. Riordan, J. Walsh, A. Murphy, and C. Ryan, "Sensor technology in autonomous vehicles : A review," *2018 29th Irish Signals and Systems Conference (ISSC)*, pp. 1–4, 2018.

[26] "Airbags." [Online]. Available: https://www.iihs.org/topics/airbags#:~:text=Typically%2C%20a%20front%20airbag%20will,up%20to%20these%20moderate%20speeds.

[27] S. Kammel and B. Pitzer, "Lidar-based lane marker detection and mapping," in *2008 ieee intelligent vehicles symposium*. IEEE, 2008, pp. 1137–1142.

[28] ——, "Lidar-based lane marker detection and mapping," in *2008 IEEE Intelligent Vehicles Symposium*, 2008, pp. 1137–1142.

[29] A. von Reyher, A. Joos, and H. Winner, "A lidar-based approach for near range lane detection," in *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, 2005, pp. 147–152.

[30] P. Lindner, E. Richter, G. Wanielik, K. Takagi, and A. Isogai, "Multi-channel lidar processing for lane detection and estimation," in *2009 12th International IEEE Conference on Intelligent Transportation Systems*, 2009, pp. 1–6.

[31] J. Hur, S.-N. Kang, and S.-W. Seo, "Multi-lane detection in urban driving environments using conditional random fields," in *2013 IEEE Intelligent Vehicles Symposium (IV)*, 2013, pp. 1297–1302.

[32] S. Zhou, Y. Jiang, J. Xi, J. Gong, G. Xiong, and H. Chen, "A novel lane detection based on geometrical model and gabor filter," in *2010 IEEE Intelligent Vehicles Symposium*. IEEE, 2010, pp. 59–64.

[33] L. Liu, X. Chen, S. Zhu, and P. Tan, "Condlanenet: a top-to-down lane detection framework based on conditional convolution," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 3773–3782.

[34] T. Zheng, Y. Huang, Y. Liu, W. Tang, Z. Yang, D. Cai, and X. He, "Clrnet: Cross layer refinement network for lane detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 898–907.

[35] J. Wang, Y. Ma, S. Huang, T. Hui, F. Wang, C. Qian, and T. Zhang, "A keypoint-based global association network for lane detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 1392–1401.

[36] L. Tabelini, R. Berriel, T. M. Paixao, C. Badue, A. F. De Souza, and T. Oliveira-Santos, "Polylanenet: Lane estimation via deep polynomial regression," in *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021, pp. 6150–6156.

[37] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[38] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[39] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[40] TuSimple, "Tusimple lane detection benchmark." [Online]. Available: https://github.com/TuSimple/tusimple-benchmark

[41] T. Sato and Q. A. Chen, "Towards driving-oriented metric for lane detection models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 17 153–17 162.

[42] G. H. Joblove and D. Greenberg, "Color spaces for computer graphics," in *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, 1978, pp. 20–25.

[43] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C.

Chen, and S. S. Iyengar, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, pp. 1–36, 2018.

[44] H. H. Tan and K. H. Lim, "Vanishing gradient mitigation with deep learning neural network optimization," in *2019 7th International Conference on Smart Computing & Communications (ICSCC)*, 2019, pp. 1–4.

[45] B. He, R. Ai, Y. Yan, and X. Lang, "Accurate and robust lane detection based on dual-view convolutional neutral network," in *2016 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2016, pp. 1041–1046.

[46] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[47] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[48] J. Kim and M. Lee, "Robust lane detection based on convolutional neural network and random sample consensus," in *International conference on neural information processing*. Springer, 2014, pp. 454–461.

[49] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[50] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[51] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition

challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[52] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[53] R. Dahyot, "Statistical hough transform," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 31, no. 8, pp. 1502–1509, 2008.

[54] I. Espejo, "Repositorio detector tradicional." [Online]. Available: https://github.com/Ign-Es/TraditionalLaneDetector

[55] ——, "Repositorio comparación de detectores." [Online]. Available: https://github.com/Ign-Es/PLN_TLD_Comparison

[56] Duckietown, "Duckietown/dt-core: This is the code that runs the core stack on the duckiebot in ros." [Online]. Available: https://github.com/duckietown/dt-core

[57] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[58] "Tensorflow." [Online]. Available: https://www.tensorflow.org/

[59] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*. PMLR, 2013, pp. 1139–1147.

[60] X. Pan, J. Shi, P. Luo, X. Wang, and X. Tang, "Spatial as deep: Spatial cnn for traffic scene understanding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[61] Y. Hou, Z. Ma, C. Liu, and C. C. Loy, "Learning lightweight lane detection cnns by self attention distillation," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1013–1021.

[62] I. Espejo, "Vídeos de entrada, visualización." [Online]. Available: https://drive.google.com/drive/folders/1-dO8DbHIGDxj6YcB56S6B49X3ZOKuzDU?usp=share_link

[63] ——, "Vídeos de entrada, visualización." [Online]. Available: https://drive.google.com/drive/folders/1a3fYmn9NcQI7MfWMYJcvHh5VHQH4NePZ?usp=share_link

[64] ——, "Rvídeos de entrada, latencia." [Online]. Available: https://drive.google.com/drive/folders/1qQX9Z76rupb-GH82nCGlVKUEuHH0BgGt?usp=share_link