



INSTITUTO TECNOLÓGICO SUPERIOR DE ATLIXCO

Organismo Público Descentralizado del Gobierno del Estado de Puebla

**Sistema de visión por computadora para la detección de las
líneas del carril y cruce, aplicado a una plataforma
autónoma móvil a escala: AutoModelCar**

OPCIÓN I.

TESIS PROFESIONAL

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO MECATRÓNICO

P R E S E N T A:
José Ángel Balbuena Palma

ASEORES: **Dra. Mariana Ibarra Bonilla**
Ing. Raúl Eusebio Grande

ATLIXCO, PUE. SEPTIEMBRE DE 2020

Resumen

En esta tesis se presenta el diseño de una arquitectura robótica, la cual sea capaz de realizar con éxito las misiones de la competencia AutoModelCar del Torneo Mexicano de Robótica (TMR). El diseño e implementación de un sistema de visión computacional, para la percepción de las líneas del carril e intersecciones de la pista de carreras del TMR. El diseño de la arquitectura es modular, con el uso de ROS (Sistema Operativo Robótico) para la integración de todos sus sistemas de visión computacional, control y actuación. El sistema de visión computacional emplea técnicas de visión clásicas y modelado matemático de las líneas carril y cruce. La implementación del sistema de visión computacional es con OpenCV y C++. Así de esta forma, optimizar los recursos del sistema embebido Jetson Nano seleccionado para realizar el computo a bordo de la plataforma robótica y no de manera externa.

Abstract

This thesis presents the design of a robotic architecture, which is capable of successfully performing the missions of the AutoModelCar competition of the Mexican Robotics Tournament (TMR). The design and implementation of a computer vision system for the perception of lane lines and intersections of the TMR race track. The architecture design is modular, with the use of ROS (Robotic Operating System) for the integration of all its computer vision, control and actuation systems. The computer vision system employs classical vision techniques and mathematical modeling of the lane and crossing lines. The implementation of the computer vision system is with OpenCV and C++. Thus, in this way, optimize the resources of the selected Jetson Nano embedded system to perform the computation on board the robotic platform and not externally.

“Sé el cambio que quieres ver en el mundo...”

Mahatma Gandhi

A mis hermanos, mi padre y especialmente a mi madre.

Agradecimientos

Quiero agradecer en primer lugar a Dios por el regalo de la vida. A mi madre por ser esa persona que siempre ha estado en todo momento para apoyarme y guiar me. A mi padre que ha hecho lo que ha podido para apoyarme. A mis dos hermanos que han sido mis compañeros de vida.

Me gustaría agradecer a la Dra. Mariana Ibarra Bonilla y al Ing. Raúl Eusebio Grande asesores de esta tesis por ser docentes que siempre se ha interesado por el aprendizaje de sus alumnos. Así como a todos mis docentes de la ingeniería que me brindaron su tiempo, conocimientos y vivencias para formar mi educación universitaria.

ÍNDICE GENERAL

Introducción	1
Capítulo 1 Protocolo	2
1.1 Antecedentes	2
1.2 Planteamiento del problema	3
1.3 Justificación	3
1.4 Limitaciones y alcances	3
1.5 Objetivos	4
1.5.1 Objetivo general.....	4
1.5.2 Objetivos particulares	4
1.6 Estado del Arte.....	4
1.6.1 Competencia Carolo-Cup	6
1.7 Hipótesis	8
1.8 Organización de la tesis.....	8
Capítulo 2 Marco teórico.....	10
2.1 Tópicos de robótica	11
2.1.1 Definición de un robot inteligente.....	11
2.1.2 Paradigmas de la robótica	11
2.1.3 Arquitecturas.....	14
2.2 ROS	15
2.2.1 ¿Qué es ROS?	15
2.2.2 ¿Entonces ROS es un sistema operativo?	16
2.1.3 Los objetivos de ROS	17
2.2.4 Componentes de ROS.....	17
2.2.5 Ecosistema de Ros	18
2.2.6 Versiones de ROS.	19
2.2.7 Conceptos de ROS	20
2.3 Visión Computacional	28
2.3.1 Representación de imágenes	28
2.3.2 Transformación a escalas de grises	29
2.3.3 Procesamiento digital de imágenes	29
2.3.4 Filtrado de imágenes	30

2.3.5 Geometría de una proyección en perspectiva 2D	32
.....	33
.....	33
2.3.5 Efecto de distorsión en perspectiva	33
2.3.5 Mapeo en perspectiva inversa	34
2.3.6 Eliminar la distorsión de la perspectiva.....	35
2.4 Extracción y modelado de datos	36
2.4.1 Histograma	36
.....	36
2.4.2 Algoritmo de la ventana deslizante	36
2.4.3 Regresión lineal y no-lineal.....	38
2.4.4 Eliminación Gaussiana	40
2.5 OpenCV	44
Capítulo 3 Metodología	46
3.1 Aspectos generales a considerar.....	47
3.2 Hardware y software	48
3.3 Diseño de la arquitectura	52
3.3 Adquisición de las imágenes.....	54
3.4 Preprocesamiento	55
3.4.1 Mapeo en perspectiva inversa.....	56
3.4.2 Conversión a escala de grises	57
3.4.3 Filtrado.....	58
3.5 Detección de las líneas del carril.....	62
3.5.1 Histograma y estimación inicial de la búsqueda	63
3.5.2 Búsqueda de los puntos de las líneas del carril.....	65
3.5.3 Regresión polinómica	68
3.5.4 Calculo del ángulo de la línea central del carril y su desviación del centro de la imagen	70
3.6 Detección de intersecciones	72
Capítulo 4 Resultados experimentales.....	77
4.1 Sección recta	79
4.2 Secciones curvas A y B	85
4.3 Sección intersección	95

4.4 Comparativa del desempeño de los modelos de las líneas del carril e intersecciones	99
4.5 Frecuencias de procesamiento de los nodos del subsistema de visión artificial.....	101
Capítulo 5 Conclusiones y trabajos futuros	102
5.1 Conclusiones.....	103
5.2 Trabajos futuros	104
Referencias	105
Apéndice A: jetson_camera.cpp.....	108
Apéndice B: lane_detection.cpp.....	110
Apéndice C: crossing_detection.cpp	122

ÍNDICE DE FIGURAS

Figura 1.1. Primera carrera de autos autónomos a escala del IPN [4].	2
Figura 1.2. 1) Tesla, 2) Waymo, 3) Spirit of Berlig y 4) RALPH [5] [6] [9] [10]......	6
Figura 1.3. Competencia Carolo-Cup [13].....	7
Figura 1.4. Modelo matemático de segundo grado de las líneas del carril [14].....	7
Figura 2.1. Paradigma reactivo.	12
Figura 2.2. Paradigma jerárquico.	13
Figura 2.3. Paradigma híbrido.	14
Figura 2.4. Multi-Comunicación de ROS [19].	16
Figura 2.5. Componentes de ROS [19].	18
Figura 2.6. Ecosistema de ROS [19].	19
Figura 2.7. Tópico con múltiples subscriptores.	22
Figura 2.8. Corriendo el nodo Maestro.....	23
Figura 2.9. Corriendo el nodo Suscriptor.....	24
Figura 2.10. Corriendo el nodo Publicador.....	24
Figura 2.11. Proporcionada información acerca del nodo publicador.....	25
Figura 2.12. Solicitud de conexión desde el nodo suscriptor.	26
Figura 2.13. Respuesta a la conexión desde el nodo publicador	26
Figura 2.14. Conexión TCPROS.	26
Figura 2.15. Tópico transmisión de mensajes.....	27
Figura 2.16. Ejemplo de comunicación de un mensaje en ROS.	28
Figura 2.17. Representación de una imagen como función $f(x, y)$	28
Figura 2.18. Procesamiento digital de imágenes [24].....	30
Figura 2.19. Filtro promedio móvil [24].	30

Figura 2.20. Modelo de una cámara estenopeica [26].	32
Figura 2.21. Fenómeno de punto de fuga [26].	33
Figura 2.22. Perspectiva 2D de la imagen de un camino [26].	33
Figura 2.23. Modelo del mapeo en perspectiva inversa [26].	34
Figura 2.24. Visualización del mapeo en perspectiva inversa [26].	35
Figura 2.25. Ejemplo de histograma.....	36
Figura 2.26. Logo de la librería OpenCV.....	45
Figura 3.1. Pista de carreras.	47
Figura 3.2. Tramo de pista para desarrollo y pruebas del sistema.	48
Figura 3.3. Vista superior de la integración.	49
Figura 3.4. Vista frontal de la integración.	50
Figura 3.5. Arquitectura del sistema tipo AutoModelCar.	53
Figura 3.6. Imagen final para publicación.....	54
Figura 3.7. Adquisición de la imagen dentro de la arquitectura.....	55
Figura 3.8. Preprocesamiento de la imagen.....	55
Figura 3.9. Área de interés.....	56
Figura 3.10. Mapeo en perspectiva inversa.....	57
Figura 3.11. Conversión a escala de grises.	57
Figura 3.12. Aplicación del filtro mediana.....	58
Figura 3.13. Segmentación por umbral.	59
Figura 3.14. Efecto del filtro de anchura blanca.	60
Figura 3.15. Proceso de la detección de las líneas del carril.	62
Figura 3.16. a) Región de interés, b) Histograma.....	63
Figura 3.17. Errores en la estimación de los puntos iniciales de la búsqueda.	64
Figura 3.18. Solución de errores en la estimación de los puntos iniciales de la búsqueda.....	65
Figura 3.19. Búsqueda de las líneas del carril por medio de algoritmo de la ventana deslizante.	66
Figura 3.20. Polinomios de segundo grado ajustadas a los puntos de la línea derecha y los puntos de la línea izquierda del carril.....	69
Figura 3.21. Línea central del carril.	70
Figura 3.22. Proceso de la detección del cruce.....	72
Figura 3.23. Rotación de la imagen preprocesada.....	73
Figura 3.24. Región de interés e histograma.....	73
Figura 3.25. Búsqueda por ventana deslizante de los puntos que pertenecen al cruce.	74
Figura 3.26. Distancia en columnas desde la línea del cruce hasta el coche.	76
Figura 4.1. Secciones de la pista de pruebas.....	78
Figura 4.2. Imágenes preprocesadas de la sección recta.	79
Figura 4.3. Posiciones en la imagen muestra de los puntos pertenecientes a las líneas izquierdas en la sección recta.....	80

Figura 4.4. Posiciones en la imagen muestra de los puntos pertenecientes a las líneas derechas en la sección recta.	81
Figura 4.5. Conjuntos de puntos agrupados que pertenecen a las líneas del carril de la sección recta.	81
Figura 4.6. Distribución de puntos agrupados de las líneas izquierdas en el tramo recto.	82
Figura 4.7. Distribución de puntos agrupados de las líneas derechas en el tramo recto.	83
Figura 4.8. Líneas obtenidas a partir de los modelos de la sección recta.	85
Figura 4.9. Imágenes preprocesadas de la sección curva A.	86
Figura 4.10. Imágenes preprocesadas de la sección curva B.	86
Figura 4.11. Posiciones en la imagen muestra de los puntos encontrados de las líneas izquierdas en la sección curva A.	87
Figura 4.12. Posiciones en la imagen muestra de los puntos de las líneas derechas en la sección curva A.	87
Figura 4.13. Posiciones en la imagen muestra de los puntos encontrados de las líneas izquierdas en la sección curva B.	88
Figura 4.14. Posiciones en la imagen muestra de los puntos encontrados de las líneas derechas en la sección curva B.	88
Figura 4.15. Conjuntos de puntos agrupados que pertenecen a las líneas del carril de la sección curva A.	89
Figura 4.16. Conjuntos de puntos agrupados que pertenecen a las líneas del carril de la sección curva B.	89
Figura 4.17. Distribución de puntos agrupados de las líneas izquierdas en la sección curva A.	91
Figura 4.18. Distribución de puntos agrupados de las líneas derechas en la sección curva A.	92
Figura 4.19. Distribución de puntos agrupados de las líneas izquierdas en la sección curva B.	92
Figura 4.20. Distribución de puntos agrupados de las líneas derechas en la sección curva B.	93
Figura 4.21. Líneas obtenidas a partir de los modelos de la sección curva A.	94
Figura 4.22. Líneas obtenidas a partir de los modelos de la sección curva B.	95
Figura 4.23. Imágenes preprocesadas de la sección de intersección.	96
Figura 4.24. Posiciones en la imagen muestra de los puntos encontrados de las líneas en la sección de intersección.	96
Figura 4.25. Conjuntos de puntos agrupados que pertenecen a las líneas de la sección de intersección.	97
Figura 4.26. Distribución de puntos agrupados de las líneas de la sección de intersección.	98

Figura 4.27. Líneas obtenidas a partir de los modelos de la sección de intersección.....	99
Figura 4.28. Comparación de errores de los modelos de las líneas del carril.	100

ÍNDICE DE TABLAS

Tabla 1. Relaciones del paradigma reactivo [18].....	12
Tabla 2. Relaciones del paradigma jerárquico [18].	13
Tabla 3. Relaciones paradigma híbrido [18].	14
Tabla 4. Tipo de datos básicos de ROS [19].....	21
Tabla 5. Búsqueda de máximos por ventana deslizante.	37
Tabla 6. Búsqueda de mínimos por ventana deslizante.	37
Tabla 7 Lista de hardware y costos.....	50
Tabla 8. Desviaciones estándares y medianas de las columnas de los puntos agrupados alrededor de las líneas del carril en el tramo recto.	81
Tabla 9. Errores de los modelos de las líneas del carril en el tramo recto.	84
Tabla 10. Desviaciones estándares y medianas de las columnas de los puntos agrupados alrededor de las líneas del carril en la sección curva A.....	90
Tabla 11. Desviaciones estándares y medianas de las columnas de los puntos agrupados alrededor de las líneas del carril en la sección curva B.....	90
Tabla 12. Errores de los modelos de las líneas del carril en la sección curva A. ..	93
Tabla 13. Errores de los modelos de las líneas del carril en la sección curva B. ..	94
Tabla 14. Desviaciones estándares y medianas de las columnas de los puntos agrupados alrededor de las líneas del carril en la sección.	97
Tabla 15. Errores de los modelos de las líneas de la sección de intersección.....	98
Tabla 16. Frecuencias de publicación de cada nodo del subsistema de visión artificial.	101

ÍNDICE DE ALGORITMOS

Algoritmo 1. Búsqueda de máximos por el método de la ventana deslizante.	37
Algoritmo 2. Búsqueda de mínimos por el método de la ventana deslizante	38
Algoritmo 3. Pseudocódigo de la eliminación gaussiana.....	43
Algoritmo 4. Filtro de anchura en dirección horizontal.....	60
Algoritmo 5. Pseudocódigo de la búsqueda de los puntos de las líneas del carril por ventana deslizante.	66
Algoritmo 6. Cálculo del ángulo de la línea central del carril y su desviación del centro de la imagen.....	71
Algoritmo 7. Cálculo del ángulo y distancia al cruce.	75

Introducción

En la actualidad, el mundo se encuentra inmerso en una carrera por el desarrollo del coche autónomo. Este desarrollo traerá consigo un impacto ambiental, social y económico nunca antes visto. De acuerdo con un estudio de Intel y Strategy Analytics, los vehículos autónomos generarán 0.8 billones de dólares para 2035 y hasta 7 billones de dólares en 2050 [1]. Sin embargo, uno de los mayores impactos en la vida humana, es si todos los vehículos fueran autónomos se eliminaría el factor humano, cuyos despistes están presentes en el 80% de los accidentes [2].

Una función fundamental en el control de vehículos autónomos es la percepción del camino. Para lograrlo se pueden combinar diversos sensores, como lo son las cámaras, LIDAR, IMU y GPS, haciendo uso de técnicas de cartografía, visión artificial e inteligencia artificial. Sin embargo, el alto costo del desarrollo de estos vehículos de tamaño real para la investigación, obliga al uso de vehículos a escala, donde se pueden probar y validar el desarrollo de sistemas de conducción autónoma, como es el caso de los sistemas de visión artificial para la detección del carril.

Este trabajo presenta una plataforma robótica tipo AutoModelCar, con un sistema de visión por computadora, para detectar las líneas del carril e intersecciones de la pista de competencia en la categoría AutoModelCar del Torneo Mexicano de Robótica. La plataforma propuesta utiliza la integración modular por medio de ROS (Sistema Operativo Robótico). El sistema de visión artificial está implementado en C++ y haciendo uso de la librería OpenCV. El sistema corre sobre una tarjeta embebida Jetson Nano de NVIDIA, que ejecuta una distribución GNU / Linux, Ubuntu 18.04. El sistema de visión artificial aplica el procesamiento de imágenes, mapeo de perspectiva inversa, extracción de características y modelado de líneas por regresión polinómica.

Capítulo 1 Protocolo

1.1 Antecedentes

En México, el Torneo Mexicano de Robótica en sus ediciones 2017, 2018 y 2019, llevo a cabo la categoría AutoModelCar, cuyo antecedente es el Proyecto “Visiones de movilidad urbana” del año dual México-Alemania 2016-2017, promovido por el Dr. Raúl Rojas, para incentivar la formación de recursos humanos en el área de investigación de los coches autónomos, realizando la entrega de carros autónomos a escala a los institutos y universidades más importantes del país [3]. Estos vehículos fueron puestos a prueba en la competencia “Primera carrera de autos autónomos a escala”, realizada en el Instituto Politécnico Nacional en el año 2017 [4]. En esta competencia, se ponen a prueba las capacidades de conducción autónoma de un vehículo a escala dentro de un entorno controlado (ver figura 1.1). Los vehículos presentados en este tipo de competición, utilizan principalmente cámaras montadas en el vehículo, para detectar las líneas del carril y las intersecciones de la pista y aplicando algunos pasos comunes: procesamiento de imágenes, extracción de características y modelado de líneas de carril.



Figura 1.1. Primera carrera de autos autónomos a escala del IPN [4].

1.2 Planteamiento del problema

El Instituto Tecnológico Superior de Atlixco no cuenta con coches autónomos a escala como lo son las plataformas AutoModelCar. Estas plataformas permiten a los estudiantes, el aprendizaje e investigación de tecnologías en la conducción autónoma. Al no tener un carro autónomo a escala o una plataforma similar al AutoModelCar, no se puede participar en la categoría de coches autónomos a escala del Torneo Mexicano de Robótica, impidiendo a la comunidad estudiantil del Tecnológico de Atlixco, de competir en esta categoría y perdiendo de esta forma, la retroalimentación dada por investigadores con mayor experiencia en estas áreas tecnológicas. Cabe resaltar que en la última edición del Torneo Mexicano de Robótica 2019, en la categoría AutoModelCar la participación de los equipos poblanos fue nula.

1.3 Justificación

Se busca desarrollar una plataforma similar al AutoModelCar, en una versión accesible tanto técnica como económicamente, para cualquier alumno de ingeniería del Instituto Tecnológico Superior de Atlixco e instituciones similares del estado de Puebla, que se encuentren interesados en el desarrollo e investigación de estas tecnologías. De esta forma facilitar e incentivar su participación en estas competencias de alto nivel tecnológico. Para lograr potenciar las habilidades de los estudiantes en áreas como robótica, visión computacional y control en aplicaciones de conducción autónoma. Esta plataforma será un primer acercamiento a estas tecnologías de vanguardia y un punto de partida para futuras mejoras tanto en hardware como en software.

1.4 Limitaciones y alcances

Debido a la amplitud de temas que abarca este trabajo y el tiempo disponible para su realización, este proyecto se dividió en dos tesis, que en conjunto abarcarán el diseño e implementación completa de un coche autónomo a escala que sea capaz de realizar las pruebas de la competencia AutoModelCar del Torneo Mexicano de

Robótica, en esta tesis nos delimitaremos al diseño de la arquitectura del coche autónomo a escala y la implementación de un sistema de visión computacional para la percepción de la pista.

1.5 Objetivos

1.5.1 Objetivo general

Diseñar y desarrollar el sistema de visión artificial para una plataforma móvil tipo AutoModelCar, capaz de detectar la pista de carreras de la categoría de coches autónomos del Torneo Mexicano de Robótica (TMR).

1.5.2 Objetivos particulares

1. Diseñar una arquitectura para la implementación de una plataforma tipo AutoModelCar, usando un sistema de integración modular como ROS (Robot Operative System), la cual sea capaz de ejecutar las misiones de la competencia coches autónomos a escala del TMR.
2. Diseñar e implementar el sistema de visión artificial capaz de detectar las líneas que delimitan el carril y las líneas de cruces de la pista, con OpenCV como librería de visión computacional.
3. Aplicar técnicas de visión por computadora y modelado matemático, para la detección de las líneas que delimitan el carril y las líneas de cruces de la pista.
4. Realizar las pruebas de validación del desempeño del sistema de visión.

1.6 Estado del Arte

Existen diversos sistemas de conducción autónoma que ya se han implementado. Uno de los más conocidos es el de Tesla con el sistema Autopiloto, el cual emplea técnicas de aprendizaje profundo por supervisión aplicadas a la visión computacional. El aprendizaje del sistema se realiza por medio de una retroalimentación de los usuarios que cuentan con un coche tesla. Este sistema

emplea sensores como cámaras, radares, ultrasónicos y GPS como sensores principales y eliminando el lidar de sus sistemas para reducir costes de construcción [5]. Waymo, es una empresa creada por Google que se enfoca en la construcción de coches autónomos. Su tecnología se basa en técnicas de inteligencia artificial como aprendizaje profundo y aprendizaje por reforzamiento, siendo la simulación la principal forma del entrenamiento de sus sistemas de inteligencia artificial. El sistema emplea cámaras, radares, GPS y específicamente el lidar como el sensor principal de su plataforma [6]. El sistema Spirit of Berlig, dirigido por el Dr. Raúl Rojas creado para la competencia DARPA Urban Challenge, logró conducir por las calles de Berlín de manera autónoma. El cual integró un sistema cámaras estéreo, lidar y GPS como principales sensores. Su sistema emplea técnicas de visión computacional basadas en modelos geométricos y en fusión con un lidar lograron mantener el automóvil en el camino que transitaba [7]. RALPH (Rapidly Adapting Lateral Position Handler), es un sistema creado por la universidad Carnegie Mellón que logró ir de la ciudad de Pittsburgh a San Diego en Estados Unidos, el sistema integró una cámara para la detección y seguimiento del camino, empleando técnicas de visión artificial logró describir la curvatura de las líneas del camino [8]. Ver sistemas de conducción autónoma en la figura 1.2.

Estos sistemas abordan el problema de la detección del carril de dos maneras. Los primeros y más avanzados (Waymo y Tesla), emplean sistemas de visión computacional basados en técnicas de inteligencia artificial como redes neuronales, técnicas de aprendizaje profundo supervisado y por reforzamiento entre otras más para lograr la percepción del camino. Sin embargo, para poder llevar a cabo la implementación de estos sistemas se requieren masivas cantidades de datos y procesamiento. La segunda forma de abordar este problema es por medio de sistemas de visión basados en modelos geométricos, como es el caso del sistema Spirit of Berlig y el proyecto RALPH, estos sistemas generan un modelo geométrico del camino a partir de imágenes obtenidas por la cámara, estos sistemas no requieren masivas cantidades de datos y procesamiento.



Figura 1.2. 1) Tesla, 2) Waymo, 3) Spirit of Berlig y 4) RALPH [5] [6] [9] [10].

1.6.1 Competencia Carolo-Cup

Los precursores más importantes y aproximados a los AutoModelCar del Torneo Mexicano de Robótica (TMR), son los prototipos de los equipos participantes en la competencia Carolo-Cup en Alemania (ver figura 1.3). A continuación, se enlistan las principales etapas que varios equipos de la competencia alemana Carolo-Cup adoptan para la detección de líneas e intersecciones de la pista:

- Preprocesamiento de la imagen: En esta etapa se aplican técnicas de visión artificial, como recorte del área de interés, mapeo de perspectiva inversa y conversión a escala de grises, así como la aplicación de filtros para la eliminación del ruido de la imagen como filtros gaussianos, median blur y Sobel.
- Segmentación de la imagen: Consiste en obtener características específicas de la imagen, como son los bordes en las imágenes por medio de los filtros Canny y binary threshold [11].
- Modelado: Para el modelado de las líneas, se emplean técnicas como la transformada de Hough [11], la cual calcula los parámetros de un modelo lineal por medio de puntos presentes en una imagen pre-procesada, o el

método iterativo RANSAC [12], el cual calcula parámetros de un modelo matemático a partir de un conjunto de datos, cuyos valores son atípicos.



Figura 1.3. Competencia Carolo-Cup [13].

Otro método para la extracción de puntos pertenecientes a las líneas del carril, es el algoritmo de la búsqueda por ventana deslizante con promedio móvil, cuyo objetivo es lograr la agrupación de los puntos pertenecientes a las líneas del carril [14]. Una vez obtenidos los puntos de las líneas del carril, se genera un modelo matemático por medio del uso de una regresión polinómica de segundo grado, sobre el conjunto de puntos agrupados (ver figura 1.4).

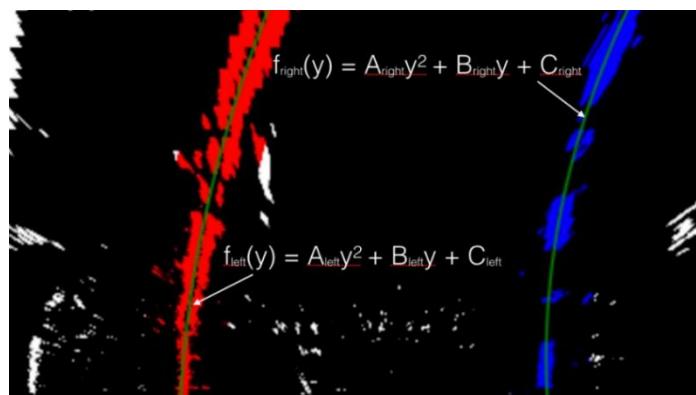


Figura 1.4. Modelo matemático de segundo grado de las líneas del carril [14].

Tomando como referencia al equipo KITcar (Campeón de la Carolo-Cup 2020) para la descripción de las principales herramientas empleadas para el desarrollo de una plataforma robótica de esta naturaleza [15], su sistema toma como base ROS (Robot Operating System) como un sistema de integración modular. Emplean OpenCV como librería de visión computacional, debido a que es de código abierto [16]. Usan las librerías Eige para la resolución de sistemas lineales y Ceres Solver para la resolución de mínimos cuadrados no lineales. Su software está desarrollado en C++ por el gran manejo de recursos y alta velocidad de ejecución que brinda este lenguaje [17].

1.7 Hipótesis

Haciendo uso de las técnicas de visión computacional clásicas y modelos matemáticos como polinomios de segundo y primer grado, es posible implementar un sistema de visión artificial robusto y rápido. Capaz de percibir la pista de carreras en la categoría AutoModelCar del TMR con pequeños errores.

1.8 Organización de la tesis

Esta sección describe de manera breve el contenido y organización del documento realizado por capítulos:

- **Capítulo 1:** Introducción. Este capítulo muestra los antecedentes y motivaciones de la tesis. El planteamiento de problema. La justificación de la tesis. Las limitaciones y alcances presentadas. Los objetivos generales y particulares del proyecto. El estado del arte de la tesis y las hipótesis de investigación planteadas.
- **Capítulo 2:** Marco teórico. Este capítulo proporciona una base teórica, que sustenta este trabajo de investigación.
- **Capítulo 3:** Metodología. Este capítulo analiza los aspectos generales a considerar para el diseño del prototipo tipo AutoModelCar, los elementos tanto de hardware y software a emplear en el prototipo y describe la metodología empleada para el diseño de la arquitectura y la implementación

de los subsistemas de visión artificial encargados de la detección de las líneas del carril y las intersecciones en la pista.

- **Capítulo 4:** Resultados experimentales. En este capítulo se analizan los resultados experimentales obtenidos de la implementación del subsistema de visión artificial para la percepción de las líneas del carril e intersecciones en la pista de pruebas.
- **Capítulo 5:** Conclusiones y trabajos futuros. Este capítulo muestra conclusiones obtenidas a partir del diseño de la arquitectura del sistema tipo AutoModelCar, así como el diseño e implementación del subsistema de visión artificial. También se proponen algunas mejoras a realizar para la continuación de este proyecto.

Capítulo 2 Marco teórico

El objetivo de este trabajo es el de diseñar la arquitectura de un sistema tipo AutoModelCar e implementar un subsistema de visión artificial, para la percepción de la pista de carreras en categoría de coches autónomos a escala del Torneo Mexicano de Robótica, por lo que lograrlo es necesario una base teórica, que sustente este trabajo de investigación. Este capítulo está dividido en cinco secciones:

- Tópicos de robótica.
- ROS (Robot Operating System)
- Visión Computacional.
- Extracción y modelado de datos.
- OpenCV.

2.1 Tópicos de robótica

2.1.1 Definición de un robot inteligente

Una criatura mecánica que puede funcionar de forma autónoma [18]. La funcionalidad autónoma se refiere a que el robot puede operar auto contenido en condiciones razonables sin necesidad de recurrir a un operador humano. La autonomía incluye que el robot puede adaptarse a cambios en el ambiente y continuar hacia su meta.

2.1.2 Paradigmas de la robótica

Un paradigma es una filosofía o un conjunto de supuestos o técnicas que caracterizan un enfoque a una clase de problemas [18]. Ningún paradigma es correcto, más bien, algunos problemas parecen más adecuados para diferentes enfoques. Aplicar el paradigma correcto hace que resolver el problema sea más sencillo. Tradicionalmente hay tres principales paradigmas para abordar el problema de controlar el comportamiento de un robot:

- Paradigma reactivo.
- Paradigma jerárquico.
- Paradigma híbrido.

Los tres paradigmas pueden ser descritos de dos formas:

- Por las relaciones entre las 3 primitivas comúnmente aceptadas por la robótica.
- Por la forma en que los datos de los sensores son procesados y distribuidos a través del sistema.

Los 3 paradigmas difieren en la forma en que están organizadas las tres primitiva comúnmente aceptadas de la robótica las cuales son:

- Percepción: Toma información de los sensores del robot y produce una salida para otras funciones.

- Planificación: Toma información desde la percepción o desde un modelo del mundo y produce una tarea para el robot.
- Actuación: Puede o no tomar las tareas provenientes desde la planificación y produce unos comandos de salida para los actuadores del robot.

El paradigma reactivo

En este paradigma el robot solo cuenta con dos primitivas la percepción y la actuación ver imagen 2.1. Las relaciones establecidas en este tipo de paradigma se observan en la tabla 1 siguiendo el flujo de las flechas.

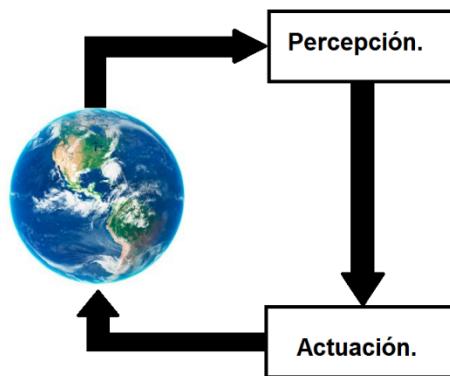


Figura 2.1. Paradigma reactivo.

Tabla 1. Relaciones del paradigma reactivo [18].

Primitivas del robot.	Entradas	Salidas
Percepción.	Datos de los sensores.	Información de los sensores.
Actuación.	Información de los sensores.	Comandos para actuadores.

Paradigma jerárquico

En este paradigma el robot cuenta con las tres primitivas la percepción, la planificación y la actuación ver imagen 2.2. Las relaciones establecidas en este paradigma entre las tres primitivas son jerárquicas, una primitiva precediendo a la siguiente primitiva, se puede observar este comportamiento en la tabla 2 siguiendo el flujo de las flechas.

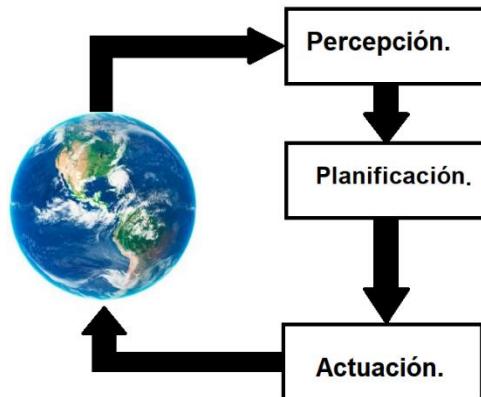


Figura 2.2. Paradigma jerárquico.

Tabla 2. Relaciones del paradigma jerárquico [18].

Primitivas del robot.	Entradas	Salidas
Percepción.	Datos de los sensores.	Información de los sensores.
Planificación.	Información (sensores y cognitiva).	Directivas.
Actuación.	Directivas.	Comandos para actuadores.

Paradigma híbrido

Este paradigma es una combinación de los dos paradigmas anteriores (ver imagen 2.3) es un paradigma en el que el robot decide como descomponer las tareas en subtareas y luego el comportamiento percepción-actuación es el más adecuado para lograr esto. En la tabla 3 se muestran las relaciones que se establecen entre las primitivas en este para este paradigma.

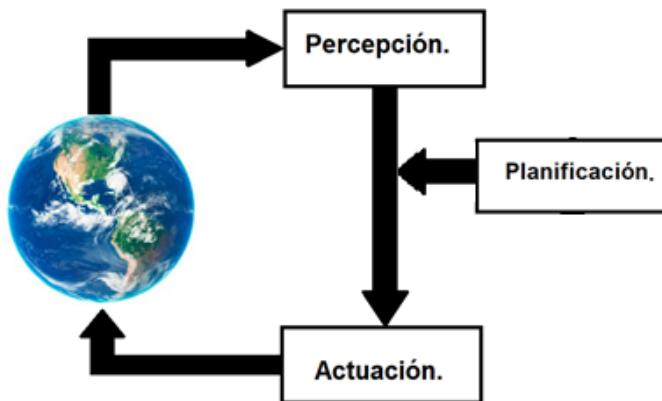


Figura 2.3. Paradigma híbrido.

Tabla 3. Relaciones paradigma híbrido [18].

Primitivas del robot.	Entradas	Salidas
Planificación.	Información (sensores y cognitiva).	Directivas.
Percepción-Actuación. (Comportamiento)	Datos de los sensores.	Comandos para actuadores.

2.1.3 Arquitecturas

Determinar el paradigma a emplear en nuestro robot es el primer paso para poder implementar la Inteligencia artificial de nuestro robot. Pero este paso es rápidamente seguido de las herramientas asociadas a este paradigma. Para visualizar como

implementar este paradigma en aplicaciones del mundo real es de mucha ayuda examinar las arquitecturas. Estas arquitecturas prevén de una plantilla para implementación de un paradigma.

¿Qué es una arquitectura?

Una arquitectura es aquella que nos proporciona los principios y forma de organizar un sistema de control [18]. También da una estructura e impone restricciones en la forma de resolver el problema. Una arquitectura describe un conjunto de componentes arquitectónicos y las interacciones entre ellos. Dado que un objetivo principal en la robótica es aprender cómo construirlos, una habilidad importante para desarrollar, es determinar si un desarrollo previo (arquitectura) se adaptara a nuestra aplicación actual. Esta habilidad ahorrar tiempo pues no reinventaremos la rueda y evitara los problemas sutiles que otras personas han encontrado y resultado. La evolución de que arquitectura implementar requiere una serie de criterios descritos a continuación:

- Soporta modularidad: ¿Muestra buenos principios de ingeniería de software?
- Focalización de nicho: ¿Qué también trabaja para la aplicación prevista?
- Facilidad de portabilidad en otros dominios: ¿Qué también funcionaria en otras aplicaciones o robots?
- Robustez: ¿Dónde es vulnerable este sistema? ¿Cómo podemos tratar de reducir esta vulnerabilidad?

2.2 ROS

2.2.1 ¿Qué es ROS?

Es un open-source, meta sistema operativo para tu robot. Esto te proporciona los servicios que tu esperarías de un sistema operativo, incluyendo abstracción de hardware, control de dispositivos de bajo nivel, implementación de funcionalidades de uso común, paso de mensajes entre procesos, y gestión de paquetes. Esto también provee herramientas y librerías para obtención, construcción, escritura y ejecución de código en múltiples plataformas y equipos.

2.2.2 ¿Entonces ROS es un sistema operativo?

Ros no es un sistema operativo como Windows, Linux y Android, este es un meta sistema operativo. Entonces qué es un meta sistema operativo, es un sistema que ejecuta procesos tales como programación, carga, monitoreo y manejo de errores, haciendo uso de una capa de virtualización y recursos computacionales distribuidos.

Entonces podemos decir que ROS corre en los sistemas operativos convencionales tales como Android Linux y Windows [19]. Por ejemplo, para emplear ROS en una distribución de Linux tal como Ubuntu, primero deberemos instalar ROS como un programa, una vez completada la instalación, las características convencionales de este sistema operativo como sistema de gestión archivos, las interfaces de usuarios y utilidades para la programación (compiladores, ejecución de hilos, etc.) podrán ser utilizadas.

ROS no solo soporta comunicación para un solo sistema operativo, sino también con múltiples sistemas operativos, hardware y programas (ver Figura 2.4). Por lo que ROS es el adecuado para el desarrollo de robots donde se combinen diversos hardware.

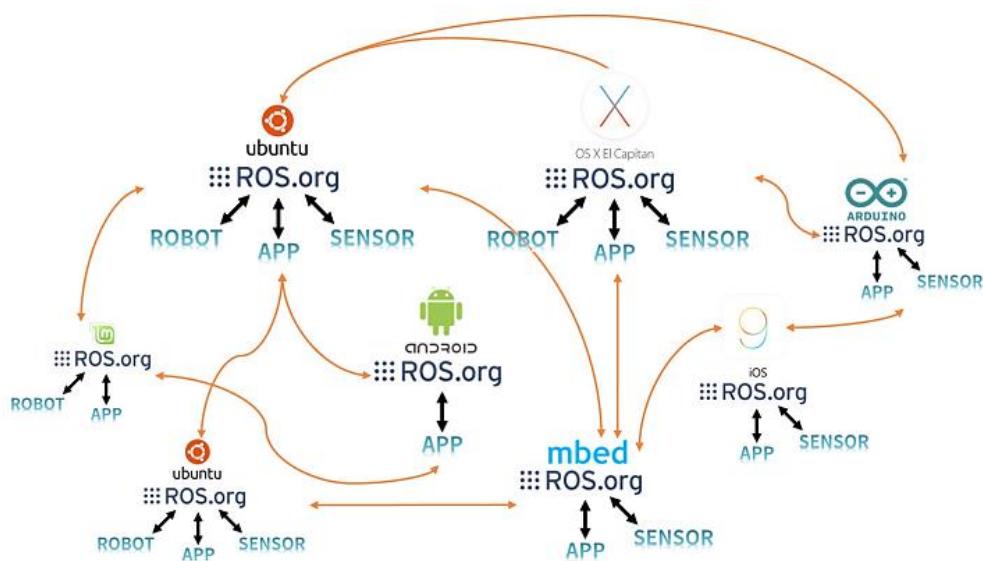


Figura 2.4. Multi-Comunicación de ROS [19].

2.1.3 Los objetivos de ROS

La meta de ROS es construir el entorno de desarrollo que permita el desarrollo software a nivel mundial, ros se enfoca en reutilización y desarrollo de código de la robótica. Las características de ROS son:

- Procesos Distribuidos: Esto es programar en mínimas unidades de procesos ejecutables (Nodos), y cada proceso correr independientemente e intercambia datos de forma sistemática.
- Gestión de Paquetes: Múltiples procesos teniendo el mismo propósito son manejados como un paquete así que es fácil de usar y desarrollar, estos paquetes se pueden compartir, modificar y distribuir.
- Repositorio Publico: Cada paquete es echo publico si el desarrollador así lo desea en repositorios como GitHub.
- API: Cuando se desarrolla un programa que usa ROS, este se diseña para llamar una API e insertar fácilmente en el código que se está implementando.
- Soporte en varios lenguajes de programación: El programa de ROS provee una librería cliente que soporta varios lenguajes, como C++, Python, Lisp, Java, etc.

2.2.4 Componentes de ROS

Ros consiste de una librería cliente que soporta varios lenguajes de programación, interfaces y hardware, comunicación para la transición y recepción de datos, es un framework para el desarrollo de aplicaciones robóticas, también ofrece herramientas con las cuales puedes controlar un robot en un espacio virtual y herramientas de desarrollo de software (ver Figura 2.5).

Development Tools.

Client Layer	roscpp	rospy	roslisp	rosjava	roslibjs
Robotics Application	Movelt!	navigatioin	executive smach	descartes	rospeex
	teleop pkgs	rocon	mapviz	people	ar track
Robotics Application Framework	dynamic reconfigure	robot localization	robot pose ekf	Industrial core	robot web tools
	tf	robot state publisher	robot model	ros control	calibration
	vision opencv	image pipeline	laser pipeline	perception pcl	laser filters
Communication Layer	common msgs	rosbag	actionlib	pluginlib	rostopic
	rosnode	roslaunch	rosparam	rosmaster	rosout
Hardware Interface Layer	camera drivers	GPS/IMU drivers	joystick drivers	range finder drivers	3d sensor drivers
	audio common	force/torque sensor drivers	power supply drivers	rosserial	ethercat drivers
Software Development Tools	RViz	rqt	wstool	rospack	catkin
	gazebo ros pkgs	stage ros			rosdep

Figura 2.5. Componentes de ROS [19].

2.2.5 Ecosistema de Ros

Un ecosistema es la estructura que une a las empresas de desarrollo de hardware, compañías que desarrollan sistemas operativos, desarrolladores de aplicaciones y usuarios [19]. Cuando ROS apareció fue lo suficientemente atractiva como para construir un ecosistema, aunque este efecto aun es pobre, el número de usuarios, empresas relacionadas a la robótica, centros de investigación, herramientas y librerías, relacionados a ROS han ido aumentando, por lo que en un futuro ROS se convertirá en un ecosistema funcional. (Ver Figura 2.6).

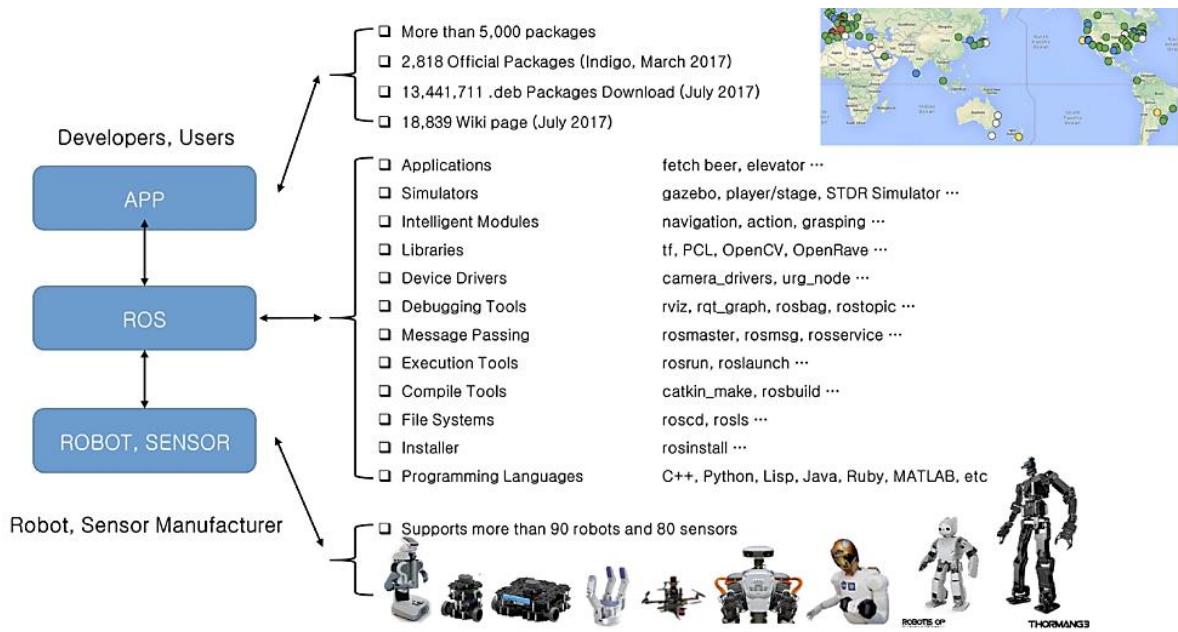


Figura 2.6. Ecosistema de ROS [19].

2.2.6 Versiones de ROS.

En noviembre de 2007 Willow Garage una empresa reconocida en el campo de los robots personales y de servicio, la misma que desarrollo la librería Poind Cloud (PCL) usada para dispositivos 3D como el Kinect y la librería para el procesamiento de imágenes de Opencv comenzó a desarrollar ROS y el 22 de enero de 2010 ROS 1.0 salió al mundo. Desde entonces las distribuciones de ROS disponibles son [19]:

- 1) ROS Box Turtle.
- 2) ROS C Turtle.
- 3) ROS Diamondback
- 4) ROS Electric Emys.
- 5) ROS Fuerte Turtle.
- 6) ROS Grooby Galapagos.
- 7) ROS Hidro Medusa.
- 8) ROS Indigo Igloo.
- 9) ROS Jade Turtle.
- 10) ROS Kinectic Frame.
- 11) ROS Lunar Loggerhead.

12) ROS Melodic Morenia.

2.2.7 Conceptos de ROS

Maestro (Master)

Es un servidor de nombres para las conexiones entre nodos y paso de mensajes [19]. El comando roscore es usado para correr el master, este registra el nombre de cada nodo y se puede obtener información cuando se necesita de cualquier nodo. La conexión entre nodos y comunicación de mensajes como son los tópicos servicios son imposibles sin el rosmaster, puesto que master se comunica con esclavos usando el protocolo XMLRPC basado en HTTP, el cual es soportado por varios lenguajes y es muy ligero, no mantiene conexión, por lo que el nodo esclavo solo se conecta cuando este necesita publicar su propia información o consultar la de alguien más.

Nodo

Un nodo es la unidad de proceso de ejecución más pequeña de ROS, se recomienda crear un nodo para cada propósito, para la fácil reutilización [19]. Los nodos registran información tal como nombre, tipo de mensaje, dirección URL, número del puerto del nodo. El registro del nodo actúa como un publicador, suscriptor, servidor de servicio y cliente de servicio basado en el registro de la información. Los nodos pueden cambiar los mensajes utilizando tópicos y servicios [20].

Paquete (Package)

Un paquete es la unidad básica de ROS [19], Las aplicaciones son desarrolladas en paquetes básicos y estos paquetes contienen un archivo para cargar otros nodos o paquete. Los paquetes también contienen todos los archivos necesarios para correr el paquete, esto incluye las librerías de dependencias de ROS para la ejecución de varios procesos, conjunto de datos y configuración de archivos [20].

Mensajes (Message)

Un nodo envía o recibe datos entre nodos por medio del mensaje [19]. Mensajes son variables como enteros, flotantes, booleanos, strings, entre otros. Un mensaje anidado es una estructura que contiene otros mensajes o arreglo de mensajes que pueden ser usados en el mensaje. Los tópicos utilizan entrega de mensajes unidireccionales [20]. A continuación, se muestra en la Tabla 4 los diferentes tipos de datos para los mensajes de ROS.

Tabla 4. Tipo de datos básicos de ROS [19].

Tipo de Dato ROS.	Serialización	Tipo de dato C++
Bool	unsigned 8-bit int	int8_t
int8	signed 8-bit int	int8_t
uint8	unsigned 8-bits int	uint8_t
int16	signed 16-bits int	int16_t
uint16	unsigned 16-bits int	uint16_t
int32	signed 32-bits int	int32_t
uint32	unsigned 32-bits int	uint32_t
int64	signed 64-bits int	int64_t
uint64	unsigned 64-bits int	uint64_t
float32	32-bit IEEE float	Float
float64	64-bit IEEE float	Doublé
String	ascii string	std::string
Time	sec/nsecs unsigned 32-bits ints	ros::time
Duration	secs/nsecs signed 32-bit ints	ros::Duration

Tópico (Topic)

Los tópicos son como los temas de una conversación [19]. El nodo publicador primero registra este tópico con el maestro y entonces comienza la publicación de

mensajes en un tópico. Los nodos subscriptores los que buscan recibir la información solicitada al tópico del nodo publicador correspondientes al nombre del tópico al que se registraron en el master. En base a esta información, el nodo suscriptor se conecta directamente al nodo publicador para intercambiar mensajes como un tópico. En la comunicación por medio de tópico, el nodo subscriptor y nodo publicador manejan el mismo tipo de dato en el mensaje. Pueden existir múltiples subscriptores para un mismo tópico como se muestra en la figura 2.7.

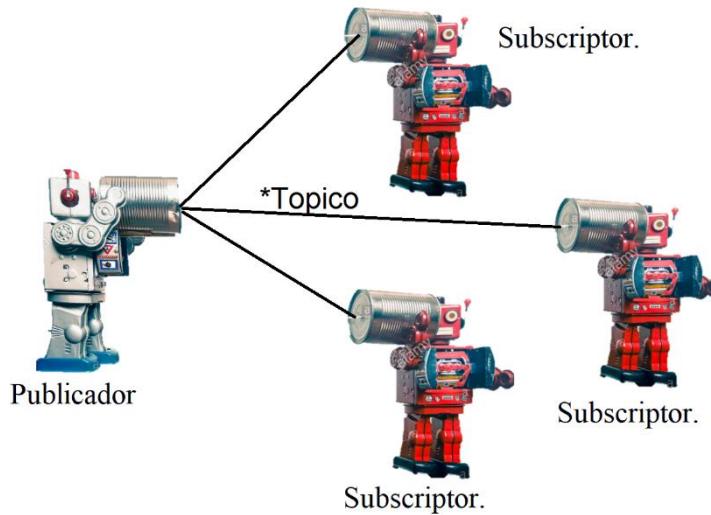


Figura 2.7. Tópico con múltiples subscriptores.

Publicar y Publicador (Publish and Publisher)

El publicar es transmitir un mensaje a un tópico correspondiente [19]. EL nodo publicador registra su información y tópico con el maestro, también envía un mensaje a todos los nodos que estén interesados en el mismo tópico. El publicador es declarado en el nodo y puede ser declarado múltiples veces en el mismo nodo [21].

Suscriptor y Suscribirse

El termino suscribirse significa la acción de recibir un mensaje correspondiente de un tópico [19]. El nodo subscriptor registra su información y tópico con el maestro, y recibe información del nodo que publica el tópico desde el maestro. Basado en la

información recibida del nodo publicador, el nodo subscriptor solicita la conexión directa con el nodo publicador y recibe mensajes del nodo publicador conectado. El suscriptor es declarado en el nodo y puede ser declarado múltiples veces en un mismo nodo [21].

La comunicación por medio de tópico es de tipo asíncrono, está basado en publicar y suscribirse, esto es muy usado para la transferencia de ciertos tipos de datos. Debido a que el tópico transmite continuamente y recibe un flujo de mensajes una vez conectado se emplea para sensores que requieren transmitir datos en un flujo continuo.

Flujo de comunicación de mensajes

El Maestro maneja la información de los nodos, y cada nodo se conecta y comunica con otros nodos como sea necesario [19]. A continuación, se muestra la secuencia para ejecutar nodos y transmitir un tópico entre ellos.

Correr el Maestro

El nodo Maestro es que maneja la información en un mensaje la comunicación entre nodos es el primer elemento que se debe ejecutar con el comando *roscore* [19]. El maestro registra el nombre de nodos, tópicos, acciones, tipo de mensajes, las direcciones URL y puertos para las conexiones entre nodos y la retransmisión de la información a petición de otros nodos (ver figura 2.8).



Figura 2.8. Corriendo el nodo Maestro.

Corriendo el nodo suscriptor

Los nodos subscriptores son cargados tanto con el comando rosrun o el comando roslaunch [19]. El nodo suscriptor registra su nombre, el nombre del tópico, el tipo de mensaje, la dirección URL y el puerto con el que el maestro está corriendo (ver figura 2.9).

```
$ rosrun nombre_paquete nombre_nodo  
$ rosrun nombre_paquete nombre_launch
```

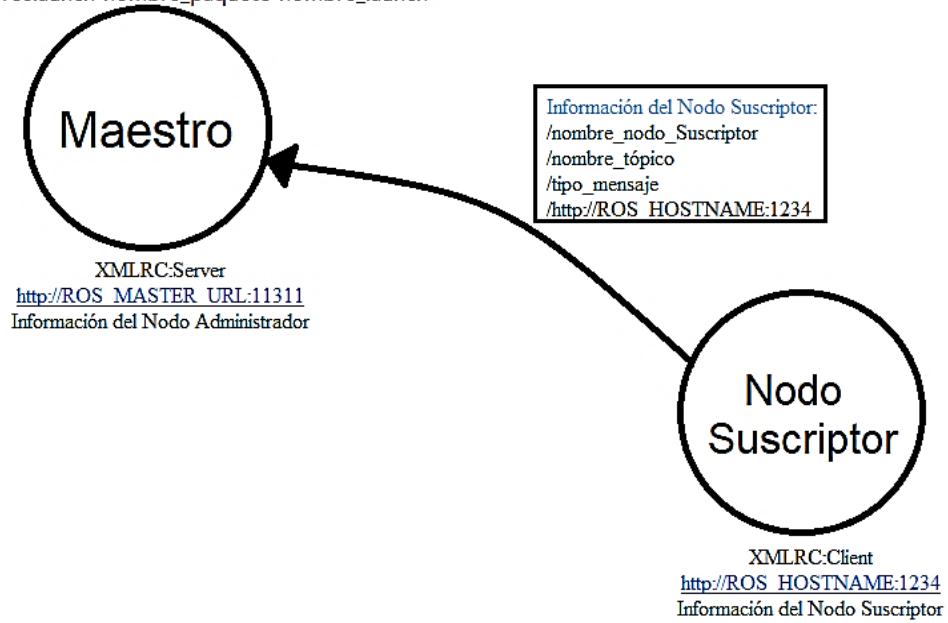
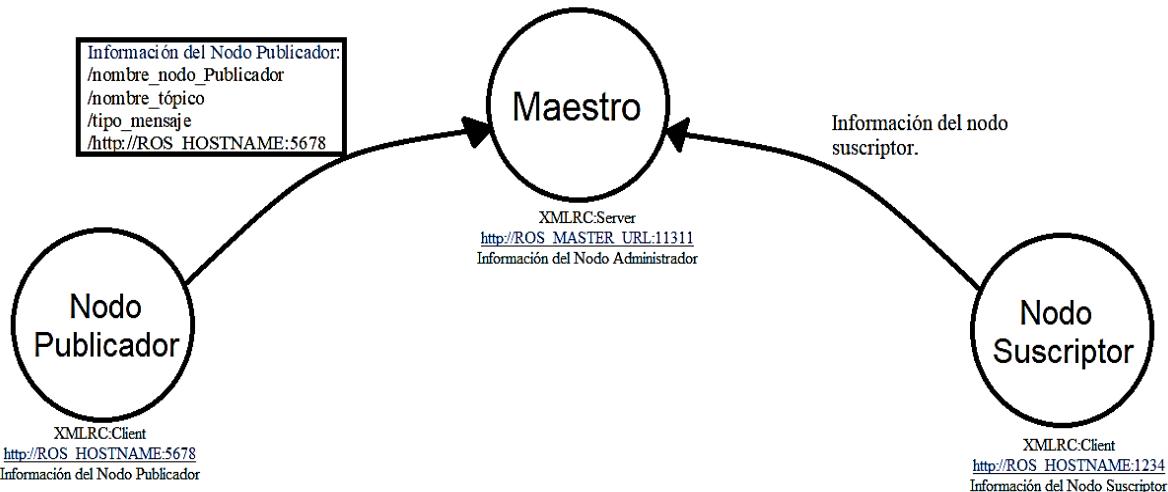


Figura 2.9. Corriendo el nodo Suscriptor.

Corriendo el nodo Publicador

El nodo publicador se puede ejecutar como el suscriptor, con los comandos rosrun y roslaunch [19]. El nodo publicador registra el nombre del nodo, el tópico del nodo, el tipo de mensaje, la dirección URL y el puerto. (ver figura 2.10).



Proporcionando la información del nodo publicador

El maestro distribuye la información como nombre del publicador, el nombre del tópico, el tipo de mensaje, la dirección URL y el número del puerto a todos los nodos suscriptores que quieren conectarse a este nodo publicador figura 2.11.

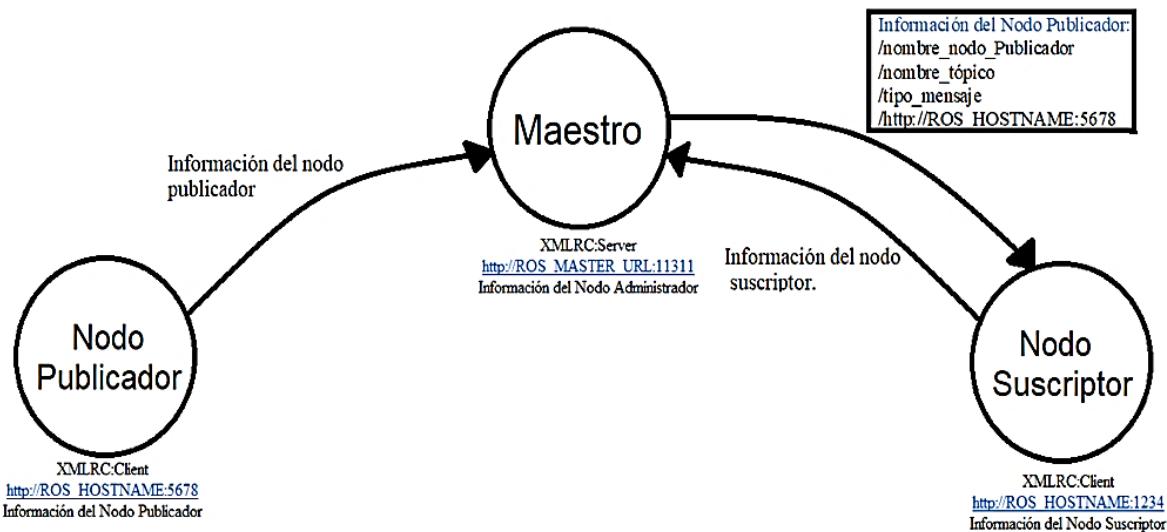


Figura 2.11. Proporcionada información acerca del nodo publicador.

Respuesta de conexión desde el nodo suscriptor

El nodo suscriptor solicita una conexión directa con el nodo publicador en base a la información recibida del maestro. Durante el proceso de solicitud, el nodo suscriptor transmite información al nodo publicador como el nombre del nodo suscriptor, el nombre del tópico y el tipo de mensaje figura 2.12.

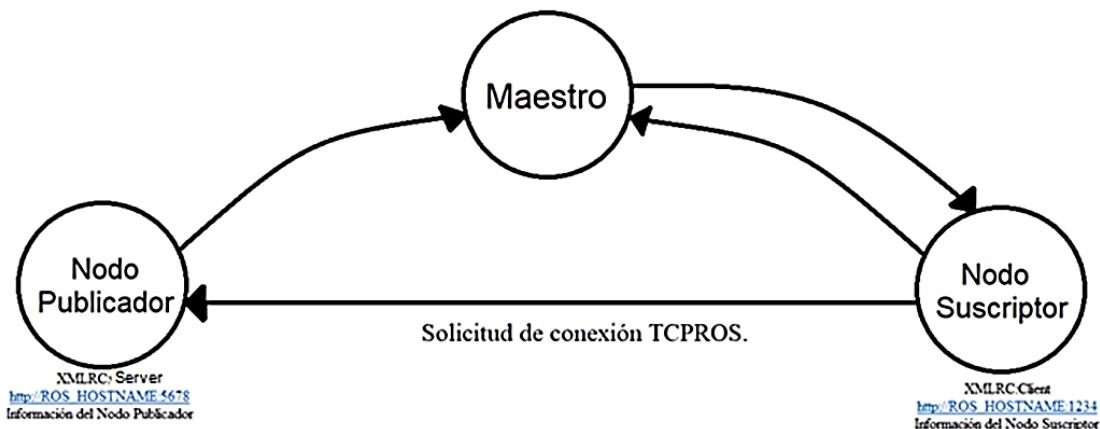


Figura 2.12. Solicitud de conexión desde el nodo suscriptor.

Respuesta de conexión desde el Nodo Publicador

El nodo publicador envía la dirección URL y el número de puerto de su servidor en respuesta a la solicitud de conexión del nodo suscriptor (ver figura 2.13).

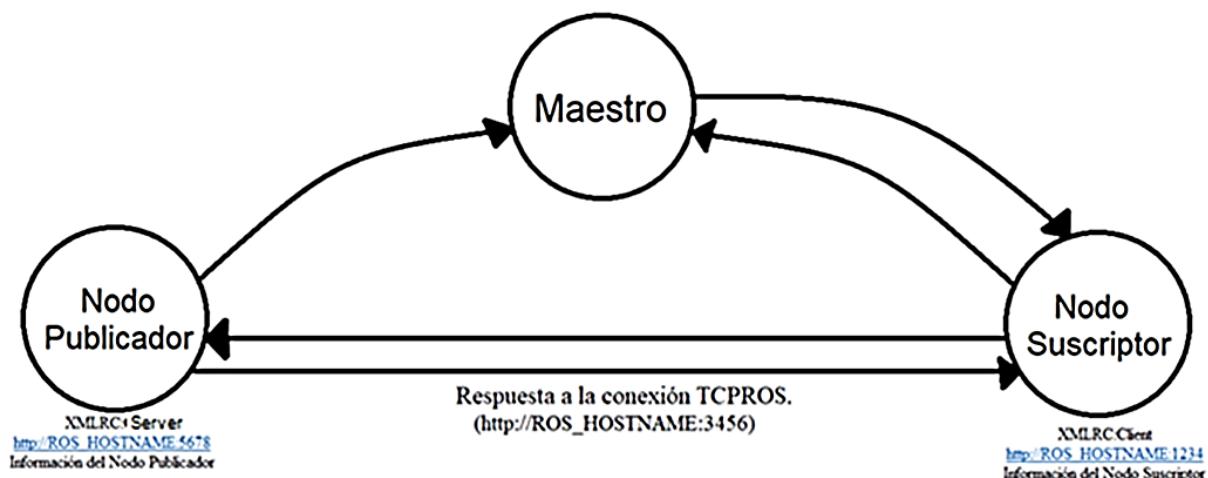


Figura 2.13. Respuesta a la conexión desde el nodo publicador.

Conexión TCPROS

El nodo suscriptor crea un cliente para el nodo publicador usando el TCPROS, y conecta al nodo publicador. En este punto la comunicación entre nodos usa un protocolo llamado TCPROS basado en TCP/IP. (ver figura 2.14).

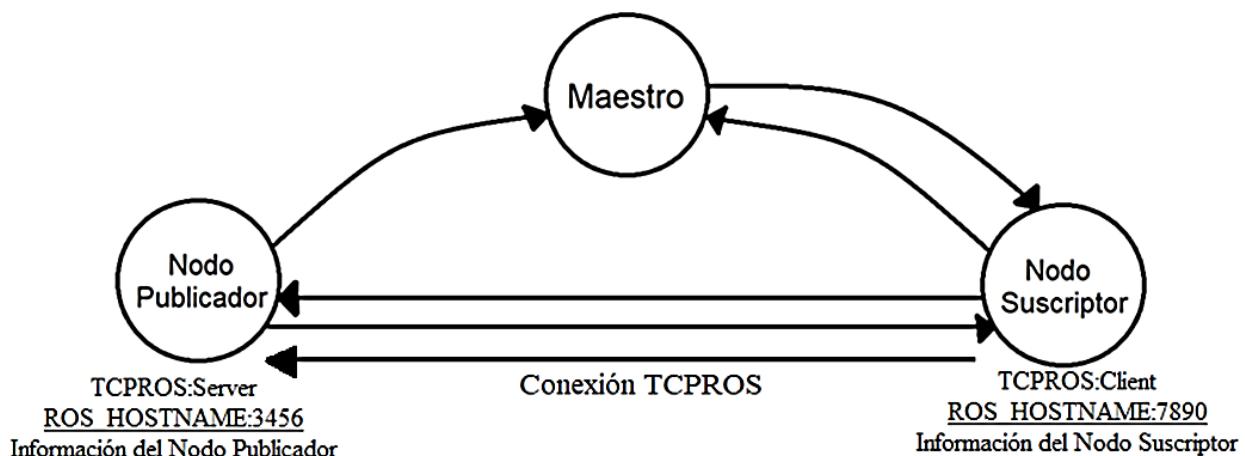


Figura 2.14. Conexión TCPROS.

Transmisión de mensajes

El nodo publicador transmite un mensaje predefinido al nodo suscriptor. La comunicación entre nodos usa el protocolo TCPROS. (ver figura 2.15) [19].

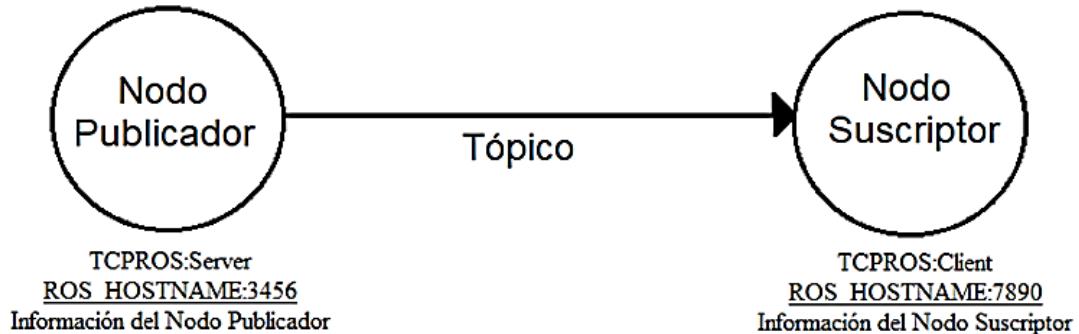


Figura 2.15. Tópico transmisión de mensajes.

A continuación, se muestra un ejemplo de cómo es el flujo de acciones para el paso de un mensaje desde el nodo publicador a un nodo suscriptor y todo dirigido por nodo. (ver figura 2.16) [19].

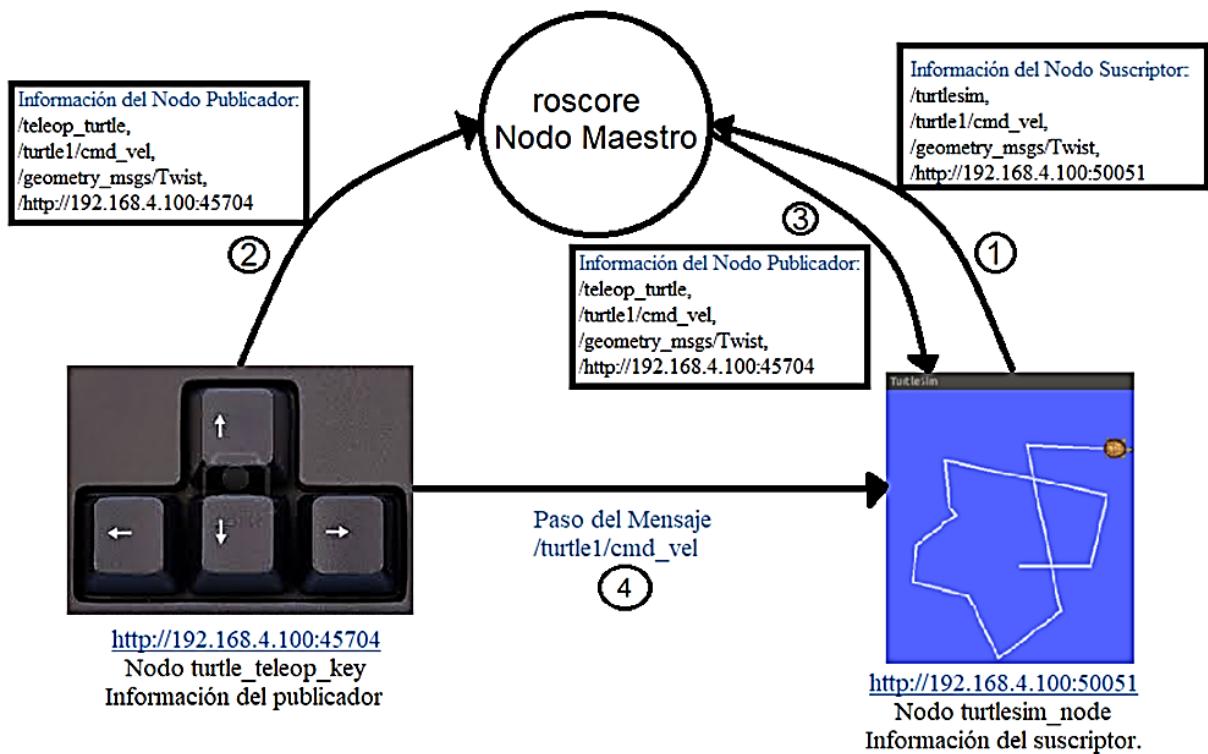


Figura 2.16. Ejemplo de comunicación de un mensaje en ROS.

2.3 Visión Computacional

2.3.1 Representación de imágenes

Una imagen digital está compuesta por pixeles como una función $f(x, y)$, cada pixel tendría un solo valor en escala de grises, cuya intensidad está dada entre 0 y 225, asumiendo que la imagen esta codificada en 8 bits, iniciando con 0 como negro y 255 para blanco [22]. Por lo que $f(x, y)$ daría la intensidad de la imagen en la posición de pixel (x, y) ver imagen 2.17, suponiendo que la función está definida sobre un rectángulo con un rango finito $F: [a, b] \times [c, d] \rightarrow [0, 255]$.

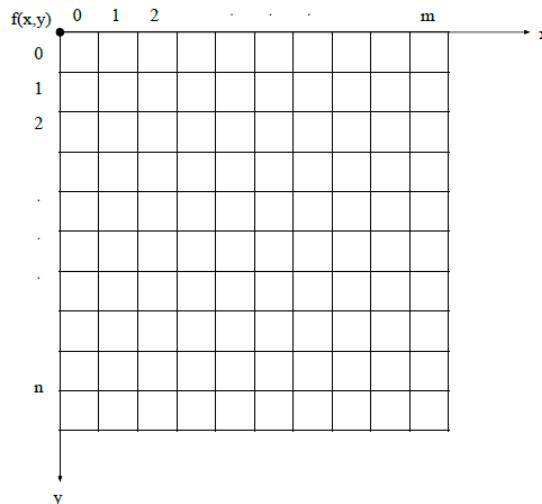


Figura 2.17. Representación de una imagen como función $f(x, y)$.

Una imagen RGB está representada como un vector de tres funciones ver ecuación 2.1, cada una de estas funciones daría la intensidad del canal de color que correspondiente. Con tres canales de color el rojo, verde y azul cada uno con valores de 0 a 255, por lo que podemos obtener un total de $256^3 = 16,777,216$ combinaciones de color.

$$f(x, y) = \begin{bmatrix} r(x, y) \\ g(x, y) \\ b(x, y) \end{bmatrix} \quad (2.1)$$

2.3.2 Transformación a escalas de grises

Para emplear ciertas técnicas de visión computacional como la obtención de contornos de imágenes es necesario un cambio de espacio de tres colores como RGB o BGR a un espacio de un único canal de color. Esta conversión implica una transformación al espacio de color en escala de grises en donde, se reduce el costo de almacenamiento de las imágenes y aumenta la capacidad de procesamiento y transferencia de estas imágenes.

La transformación de una imagen RGB a escala de grises corresponde a un mapeo o proyección de un espacio vectorial de una alta dimensión a una de menor dimensión $R^3 \rightarrow R$ [23]. La función empleada para la transformación del espacio RGB al espacio en escala de grises está dada por la ecuación 2.2.

$$g(x, y) = f(x, y, 0) \cdot 0,299 + f(x, y, 1) \cdot 0,587 + f(x, y, 2) \cdot 0,114 \quad (2.2)$$

en donde $g(x, y)$ representa la imagen en escala de grises. Esta operación se aplicada a cada píxel (x, y) de la imagen RGB, creando una imagen en escala de grises.

2.3.3 Procesamiento digital de imágenes

Hay dos tipos principales de procesamiento digital de imágenes, filtrado de imágenes y transformación geométrica de las imágenes ver imagen 2.18 [24]. La aplicación de un filtro a una imagen altera los colores de la imagen sin alterar la posición de los pixeles, mientras la aplicación de una transformación geométrica a la imagen altera la posición de los pixeles en la imagen, sin alterar el color de esta.

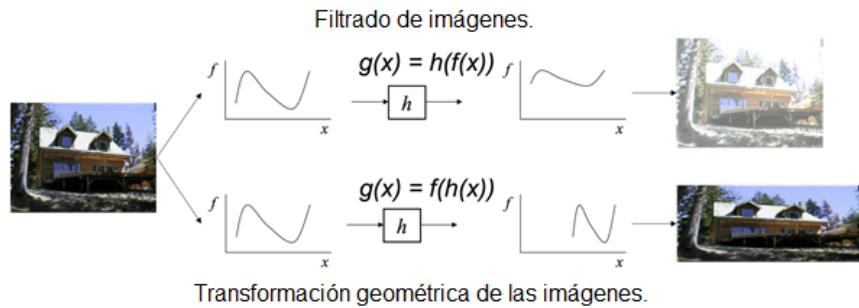


Figura 2.18. Procesamiento digital de imágenes [24].

2.3.4 Filtrado de imágenes

El objetivo de usar filtros es modificar o mejorar las propiedades de una imagen y extraer información valiosa de las imágenes como son bordes, cérneros y manchas. Otra aplicación de los filtros es eliminar ruido de la imagen, haciendo a la imagen más atractiva visualmente [24].

Las dos implementaciones de los filtros más comunes son filtro promedio móvil y el filtro de segmentación de imagen. El filtro mediano móvil remplaza cada pixel con el valor promedio del pixel y una ventana cercana de pixeles el efecto es una imagen más suave características nítidas eliminadas. En la figura 2.19 se observa la generación de una nueva imagen $G[x, y]$, a partir de la aplicación de un filtro mediana móvil con una ventana de 3x3 a la imagen $F[x, y]$.

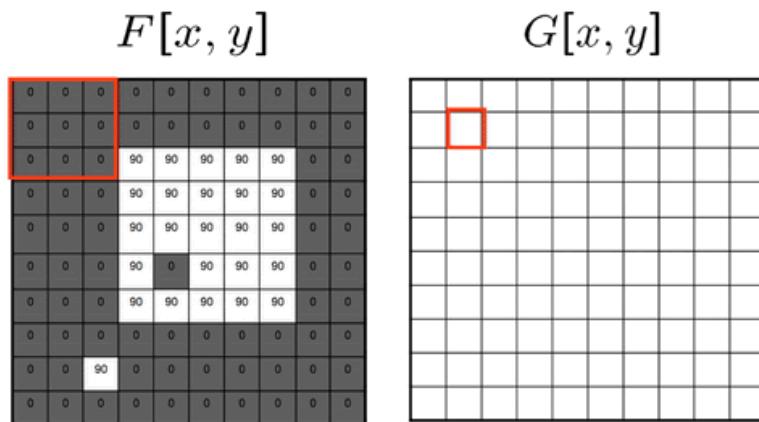


Figura 2.19. Filtro promedió móvil [24].

La segmentación de imágenes es la partición de imágenes en regiones donde los píxeles tienen atributos similares, por lo que la imagen se representa en una forma más simplificada, de esta manera nosotros podemos identificar objetos y contornos de una manera más sencilla. Hay muchas formas de hacer esta segmentación, una de las implementaciones más simples es el umbral binario que puede ser definida como la ecuación 2.3, donde cada pixel es remplazado por una intensidad de 255 (color blanco) si este es mayor que el valor de umbral y todos los otros casos se remplaza con una intensidad de 0 (color negro).

$$g[x, y] = \begin{cases} 255, & f[x, y] > \text{valor de umbral} \\ 0, & f[x, y] \leq \text{valor de umbral} \end{cases} \quad (2.3)$$

Convolución 2D

La matemática para muchos filtros puede ser expresada de manera principal usando una convolución 2D, como suavizar imágenes y detección de bordes. Una convolución en 2D opera en dos imágenes, una funciona como la entrada y la otra llamada kernel trabajando como filtro [24]. La convolución expresa la cantidad de superposición de una función a medida que se desplaza sobre otra función. Una convolución 2D se puede expresar como una función que realiza una operación de vecindad (ecuación 2.4), que se aplica a cada píxel en una imagen $f(x, y)$ generando una nueva imagen $g(x, y)$.

$$g(x, y) = \sum_{k,l} f(x + k, y + l)h(k, l) \quad (2.4)$$

en donde $h(k, l)$ es el kernel de la convolución y representa los coeficientes del filtro [25]. En la operación de filtrado para la remoción de ruido en la imagen, uno de los filtros más empleados es el filtro mediano, que realiza un suavizado a partir de la obtención de un promedio local, por lo que, cada píxel en la imagen de salida es el promedio de los píxeles en su vecindad. El kernel de este tipo de filtros está determinado por la función de la ecuación 5. El tamaño y geometría del kernel determinarán el efecto borroso sobre la imagen de salida; siendo la máscara cuadrada ($k = l$) la más comúnmente empleada.

$$h(k, l) = \frac{1}{k \cdot l} \begin{bmatrix} a_{00} & a_{01} & \dots & a_{0l} \\ a_{10} & a_{11} & \dots & a_{1l} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k0} & a_{k1} & \dots & a_{kl} \end{bmatrix} \quad (2.5)$$

2.3.5 Geometría de una proyección en perspectiva 2D

Una operación fundamental de la visión computacional, es la captura de imágenes de objetos 3D dentro de una proyección en perspectiva 2D [26]. Debido a esta proyección de perspectiva la imagen resultante, contiene distorsiones causadas por el factor de escorzo y punto de fuga. Para entender la geometría de una proyección en perspectiva 2D, se puede emplear el modelo de una cámara estenopeica ver figura 2.20. El plano de la imagen en una cámara se encuentra dentro de una distancia focal, f por detrás del centro de proyección, con la imagen de la captura del objeto en su orientación inversa.

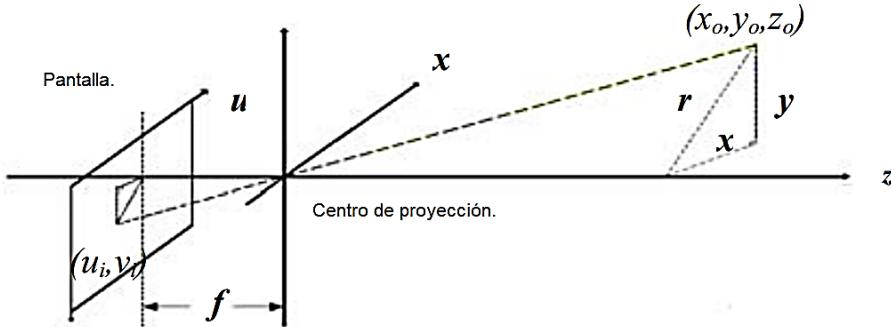


Figura 2.20. Modelo de una cámara estenopeica [26].

La posición de la imagen capturada puede ser descrita matemáticamente, tanto en términos de su componente u o utilizando los componentes u y v , con respecto a la posición actual en el espacio (vista 3D) [26]. Usando solamente la componente u la relación está dada por la ecuación 2.6.

$$\frac{u}{x} = \frac{f}{z} \quad (2.6)$$

Mientras que en términos de sus componentes u y v , está dada por la por la ecuación 2.7.

$$(u, v) = \left(f \frac{x}{z}, f \frac{y}{z} \right) \quad (2.7)$$

Retomando el modelo de la cámara estenopeica de la imagen 2.17, se puede observar que al alejarse a lo largo del eje z, como el valor de z incrementara. De la ecuación 7, podemos ver que el incremento en el valor de z causará que el objeto (de ancho u y altura v) reducirá su tamaño, este fenómeno es conocido como factor de escorzo. El punto de fuga ocurre cuando un conjunto de líneas paralelas proyectadas parece converger e interceptarse en un punto. La figura 2.21 muestra este fenómeno [26].

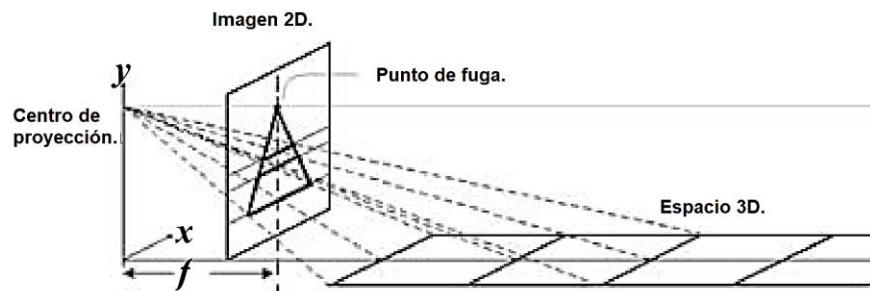


Figura 2.21. Fenómeno de punto de fuga [26].

2.3.5 Efecto de distorsión en perspectiva

En la realidad los límites de la carretera son paralelos entre sí, pero la vista en perspectiva 2D, parecen converger en un punto en común, en el punto de fuga. El efecto de proyección en perspectiva ha distorsionado los límites reales del camino. La figura 2.22 se muestra vista en perspectiva 2D de una carretera capturada con una cámara.

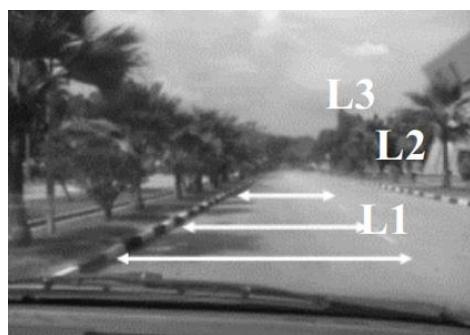


Figura 2.22. Perspectiva 2D de la imagen de un camino [26].

En la figura 2.22 se puede observar como la anchura del camino se distorsiona reduciéndose cada vez más, a medida que nos acercamos al punto de fuga. Los pixeles superiores (L3) corresponden a la distancia más lejana al vehículo, los pixeles inferiores corresponden a la distancia más cercana al vehículo.

2.3.5 Mapeo en perspectiva inversa

Como ya se mencionó hay dos tipos principales de procesamiento digital de imágenes, filtrado de imágenes y transformación geométrica de las imágenes [26]. El mapeo en perspectiva inversa es una transformación geométrica, la cual proyecta cada pixel de la vista en perspectiva 2D de un objeto 3D y lo reasigna a una nueva posición, construyendo una nueva imagen en un plano inverso. Matemáticamente el mapeo en perspectiva inversa puede describirse como una proyección desde un espacio euclíadiano $W = \{(X, Y, Z)\} \in R^3$ a un espacio en $I = \{(u, v)\} \in R^2$. Esto dará como resultado la vista de pájaro en la imagen, y, por lo tanto, eliminando el efecto de la perspectiva. En la figura 2.23 se muestra el modelo del mapeo en perspectiva inversa, del lado izquierdo se muestra la vista superior del modelo geométrico, mientras que en el lado derecho se muestra la vista lateral del modelo. De este modelo podemos obtener dos ecuaciones (ecuaciones 2.8 y 2.9).

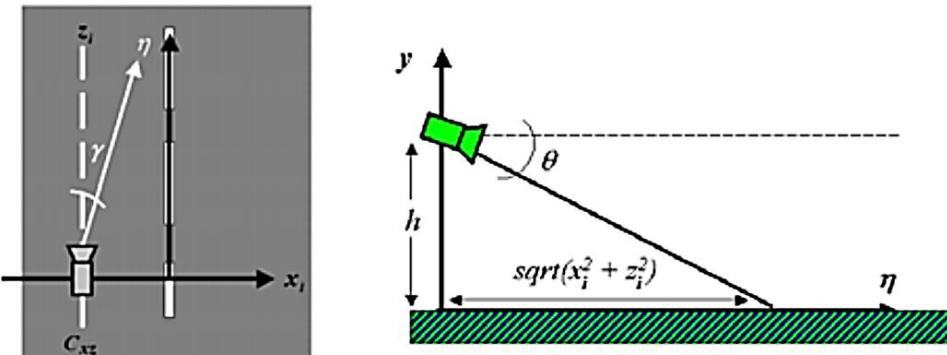


Figura 2.23. Modelo del mapeo en perspectiva inversa [26].

$$u(x, 0, z) = \frac{\gamma(x, 0, z) - (Y - a)}{\frac{2\alpha}{n - 1}} \quad (2.8)$$

$$v(x, 0, z) = \frac{\theta(x, 0, z) - (\theta - a)}{\frac{2\alpha}{m-1}} \quad (2.9)$$

Donde:

$$\gamma = \tan^{-1} \left(\frac{z}{x} \right) \quad \theta = \tan^{-1} \left(\frac{h}{\sqrt{x^2 + z^2}} \right)$$

γ es el ángulo entre la proyección del eje óptico y el plano, $y=0$. θ es el ángulo entre el eje óptico y el horizonte. α es la apertura angular de la cámara. $n \times m$ es la resolución de la imagen.

2.3.6 Eliminar la distorsión de la perspectiva

La implementación del mapeo en perspectiva inversa, asume que la superficie de la carretera en la imagen, se encuentra proyectada en el plano x-z [26]. En la figura 2.24 se muestra gráficamente la implementación del mapeo en perspectiva inversa. En la imagen en perspectiva 2D, el valor del ángulo γ cambia de negativo a positivo como el escaneo se realiza de izquierda a derecha y viceversa mientras el ángulo θ cambia verticalmente. Ambos γ y θ actúan como factores de ponderación de proyección para todos los píxeles cuando el mapeo en perspectiva inversa proyecta y reasigna la imagen en perspectiva, eliminando la distorsión presente.

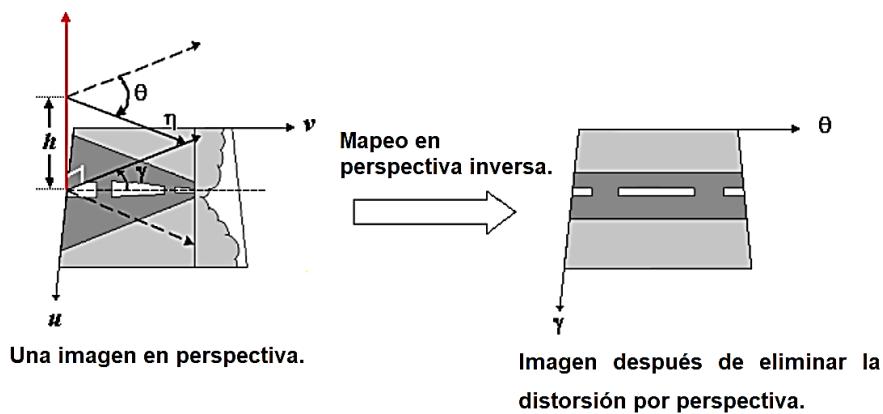


Figura 2.24. Visualización del mapeo en perspectiva inversa [26].

2.4 Extracción y modelado de datos

2.4.1 Histograma

Un histograma es usado para mostrar la distribución de valores de datos alrededor de una recta numérica real [27]. Un histograma es creado dividiendo el rango de los datos en un pequeño número de intervalos o contenedores. Se cuenta el número de observaciones que caen en cada intervalo. Esto da la distribución de frecuencias. Un histograma es una distribución de frecuencias en el eje vertical represente el conteo (frecuencia) y eje horizontal representa el posible rango de valores de datos, en la figura 2.25 se muestra un ejemplo de histograma de la suma de intensidades de color por columna de una imagen en escala de grises.

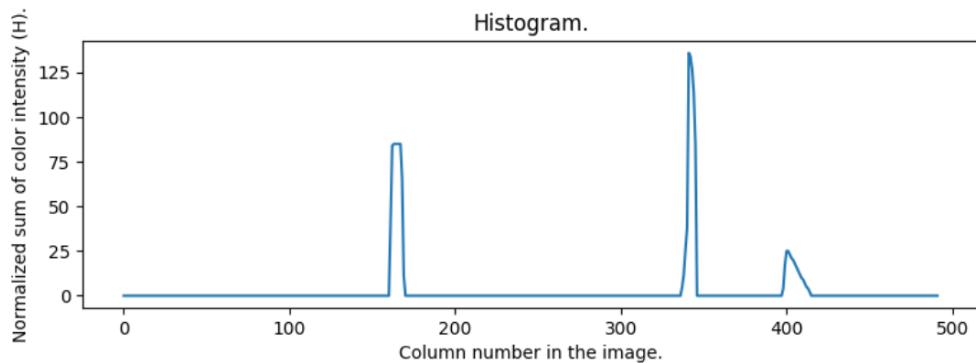


Figura 2.25. Ejemplo de histograma.

2.4.2 Algoritmo de la ventana deslizante

El algoritmo de la ventana deslizante se utiliza para realizar una operación requerida en un tamaño de ventana específico, en un arreglo o matriz [28]. La ventana comienza desde el primer elemento y sigue desplazándose por un elemento. La operación requerida puede ser encontrar los números mininos o máximo del arreglo presente en cada ventana.

Un ejemplo es buscar los máximos o mininos para un arreglo cuyos elementos son [1, 3, -1, -3, 5, 3, 6, 7] empleando el algoritmo de la ventana deslizante, con una ventana de búsqueda de 3 elementos, ver tabla 5 y tabla 6, para observar la

actuación de algoritmo. En los algoritmos 1 y 2 se muestran los pseudocódigos de búsqueda de máximos y mínimos por ventana deslizante.

Tabla 5. Búsqueda de máximos por ventana deslizante.

Posición de la ventana								Máximo
1	3	-1	-3	5	3	6	7	3
1	3	-1	-3	5	3	6	7	3
1	3	-1	-3	5	3	6	7	5
1	3	-1	-3	5	3	6	7	5
1	3	-1	-3	5	3	6	7	6
1	3	-1	-3	5	3	6	7	7

Tabla 6. Búsqueda de mínimos por ventana deslizante.

Posición de la ventana								Mínimo
1	3	-1	-3	5	3	6	7	-1
1	3	-1	-3	5	3	6	7	-3
1	3	-1	-3	5	3	6	7	-3
1	3	-1	-3	5	3	6	7	-3
1	3	-1	-3	5	3	6	7	3
1	3	-1	-3	5	3	6	7	3

Algoritmo 1. Búsqueda de máximos por el método de la ventana deslizante.

MaxSlidingWindow (a, k):
Input: a vector of size n , k window size.
Output: x vector of size $n - k + 1$
1: for i in 0 to $n - k$ do
2: $\max \leftarrow a_i$
3: for j in 0 to $k-1$ do
4: if $a_{i+j} > \max$ then
5: $\max \leftarrow a_{i+j}$
6: end if
7: $x_i \leftarrow \max$

```

8: end for
9: end for

```

Algoritmo 2. Búsqueda de mínimos por el método de la ventana deslizante

MinSlidingWindow (a, k):

Input: a vector of size n , k window size.

Output: x vector of size $n - k + 1$

```

1: for i in 0 to  $n - k$  do
2:   min  $\leftarrow a_i$ 
3:   for j in 0 to  $k-1$  do
4:     if  $a_{i+j} < \text{min}$  then
5:       min  $\leftarrow a_{i+j}$ 
6:     end if
7:    $x_i \leftarrow \text{min}$ 
8: end for
9: end for

```

2.4.3 Regresión lineal y no-lineal

La regresión lineal y la no lineal es un método numérico, que describe relaciones de un conjunto de datos experimentales [29]. Los modelos de regresión no lineal son paramétricos, donde el modelo se describe como una ecuación no lineal. El proceso de regresión es también conocido como ajuste de curvas y existen distintas técnicas empleadas para obtener dichos polinomios que se ajusten al conjunto de datos, como por ejemplo el algoritmo Gauss-Newton, el algoritmo de descenso por gradiente o el algoritmo Levenberg-Marquardt. Los parámetros pueden tomar la forma de una función exponencial, trigonométrica, de potencia o cualquier otra función no lineal. Para determinar las estimaciones de parámetros no lineales, generalmente se utiliza un algoritmo iterativo. El problema de mínimos cuadrados se define como la ecuación 2.10:

$$F(x) = \sum_{i=1}^m (f_i(x))^2 \quad (2.10)$$

en donde $f_i: R^n \rightarrow R, i = 1, \dots, m$ son funciones dadas y $m > n$. A su vez, el problema de los mínimos cuadrados es un caso especial de un problema más general: dada una función $F : R^n \rightarrow R$, encontrar un argumento de F para obtener el valor mínimo de la función de costo ecuación 2.11:

$$x^+ = \operatorname{argmin}_x \{F(x)\} \quad (2.11)$$

Dado que el problema es difícil de resolver para el caso general, sólo se presentará el caso de solución en donde se encuentra un minimizador local para F dentro de una cierta región determinada por δ ver ecuación 2.12:

$$F(x^*) \leq F(x) \quad \text{para} \quad \|x - x^*\| < \delta \quad (2.12)$$

Para el caso particular en donde $f(x) = a_0 + a_1x + a_2x^2, F(x)$ está definida como la ecuación 2.13:

$$F(x) = \sum_{i=1}^m (y_i - f(x_i))^2 \quad (2.13)$$

donde m es el número de observaciones dadas. Para minimizar $F(x)$ mediante el método por descenso de gradiente, se toma la derivada de $F(x)$ con respecto a cada coeficiente $a_k, k = 0, 1, 2$ igual a cero ver ecuaciones 2.14, 2.15 y 2.16:

$$\frac{\partial F(x)}{\partial a_0} = -2 \left(\sum_{i=1}^m (y_i - f(x_i))^2 \right) = 0 \quad (2.14)$$

$$\frac{\partial F(x)}{\partial a_1} = -2 \left(\sum_{i=1}^m (y_i - f(x_i))^2 \right) x = 0 \quad (2.15)$$

$$\frac{\partial F(x)}{\partial a_2} = -2 \left(\sum_{i=1}^m (y_i - f(x_i))^2 \right) x^2 = 0 \quad (2.16)$$

O reescribiendo como la ecuación 2.17 y 2.18:

$$\begin{bmatrix} m & \sum_{i=1}^m x_i & \sum_{i=1}^m x_i^2 \\ \sum_{i=1}^m x_i & \sum_{i=1}^m x_i^2 & \sum_{i=1}^m x_i^3 \\ \sum_{i=1}^m x_i^2 & \sum_{i=1}^m x_i^3 & \sum_{i=1}^m x_i^4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^m y_i \\ \sum_{i=1}^m y_i * x_i \\ \sum_{i=1}^m y_i * x_i^2 \end{bmatrix} \quad (2.17)$$

$$B = \begin{bmatrix} m & \sum_{i=1}^m x_i & \sum_{i=1}^m x_i^2 \\ \sum_{i=1}^m x_i & \sum_{i=1}^m x_i^2 & \sum_{i=1}^m x_i^3 \\ \sum_{i=1}^m x_i^2 & \sum_{i=1}^m x_i^3 & \sum_{i=1}^m x_i^4 \end{bmatrix}, A = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}, X = \begin{bmatrix} \sum_{i=1}^m y_i \\ \sum_{i=1}^m y_i * x_i \\ \sum_{i=1}^m y_i * x_i^2 \end{bmatrix} \quad (2.18)$$

Por tanto, para encontrar el vector A de coeficientes que minimizan el error cuadrático medio, se debe resolver B-1 por eliminación Gaussiana, obteniendo un sistema de ecuaciones como en la ecuación 2.19:

$$A = B^{-1}X \quad (2.19)$$

2.4.4 Eliminación Gaussiana

Es un algoritmo que sirve para resolver sistemas de ecuaciones líneas [29]. Donde se eliminan incógnitas y se sustituyen hacia atrás. Este método está diseñado para resolver un sistema general de n ecuaciones ver ecuaciones 2.20:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \quad (2.20 \text{ a})$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n = b_2 \quad (2.20 \text{ b})$$

$$\vdots \qquad \vdots$$

$$a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n = b_n \quad (2.20 \text{ c})$$

Eliminación hacia delante de incógnitas

La primera fase consiste en reducir el conjunto de ecuaciones a un sistema triangular superior. El paso inicial es eliminar la primera incógnita x_1 , desde la segunda hasta la n-ésima ecuación. Se multiplica la ecuación (2.20 a) por a_{21}/a_{11} para obtener:

$$a_{21}x_1 + \frac{a_{21}}{a_{11}}a_{12}x_2 + \cdots + \frac{a_{23}}{a_{11}}a_{1n}x_n = \frac{a_{21}}{a_{11}}b_1 \quad (2.21)$$

Esta ecuación se resta de la ecuación (2.20 b) para dar:

$$\left(a_{22} - \frac{a_{21}}{a_{11}}a_{12}\right)x_2 + \cdots + \left(a_{2n} - \frac{a_{21}}{a_{11}}a_{1n}\right)x_n = b_2 - \frac{a_{21}}{a_{11}}b_1$$

O

$$a_{22}'x_2 + \cdots + a_{2n}'x_n = b_2'$$

Donde el superíndice indica que los elementos han cambiado sus valores originales. Este procedimiento se repite para todas las ecuaciones restantes. Por ejemplo, la ecuación (2.20 a) se puede multiplicar por a_{31}/a_{11} y el resultado se resta de la tercera ecuación. Se repite el procedimiento a las ecuaciones restantes y se obtiene como resultado el sistema:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \quad (2.22 \text{ a})$$

$$a'_{22}x_2 + a'_{23}x_3 + \cdots + a'_{2n}x_n = b'_2 \quad (2.22 \text{ b})$$

$$a'_{32}x_2 + a'_{33}x_3 + \cdots + a'_{3n}x_n = b'_3 \quad (2.22 \text{ c})$$

$$\vdots \qquad \vdots$$

$$a'_{n2}x_2 + a'_{n3}x_3 + \cdots + a'_{nn}x_n = b'_n \quad (2.22 \text{ d})$$

En el procedimiento anterior la ecuación (2.20 a) recibe el nombre de ecuación pivote, y a_{11} se denomina elemento pivote. El proceso de multiplicación del primer renglón por a_{21}/a_{11} es equivalente a dividirla entre a_{11} y multiplicarla por a_{21} , esta operación se conoce como normalización. A continuación, se repite el procedimiento para eliminar la segunda incógnita en las ecuaciones (2.22c) hasta (2.21 d). Para lograr esto se multiplica la ecuación (2.22 b) por a'_{32}/a'_{22} y se resta el resultado de la ecuación (2.22 c). Se realiza esto de forma similar en las ecuaciones restantes obteniendo el siguiente sistema:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1$$

$$a'_{22}x_2 + a'_{23}x_3 + \cdots + a'_{2n}x_n = b'_2$$

$$a''_{33}x_3 + \cdots + a''_{3n}x_n = b''_3$$

$$\vdots \quad \vdots$$

$$a''_{n3}x_3 + \cdots + a''_{nn}x_n = b''_n$$

Donde el superíndice biprima señala que los coeficientes se han modificado dos veces. Este procedimiento continúa usando las ecuaciones pivote restantes. La última manipulación en esta secuencia es el uso de la $(n-1)$ -ésima ecuación para eliminar el término x_{n-1} de la n -ésima ecuación. Obteniendo un sistema triangular superior.

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \quad (2.23 \text{ a})$$

$$a'_{22}x_2 + a'_{23}x_3 + \cdots + a'_{2n}x_n = b'_2 \quad (2.23 \text{ b})$$

$$a''_{33}x_3 + \cdots + a''_{3n}x_n = b''_3 \quad (2.23 \text{ c})$$

$$\vdots \quad \vdots$$

$$a^{(n-1)}_{nn}x_n = b^{(n-1)}_n \quad (2.23 \text{ d})$$

Sustitución hacia atrás.

De la ecuación (2.23 d) se despeja x_n :

$$x_n = \frac{b^{(n-1)}_n}{a^{(n-1)}_{nn}} \quad (2.24)$$

De este resultado se despeja hacia atrás en la (n-1) n-ésima ecuación y despejar x_{n-1} . Este procedimiento se repite para evaluar las x restantes, se representa mediante la ecuación 2.25:

$$x_i = \frac{b^{(i-1)}_i - \sum_{j=i+1}^n a^{(i-1)}_{ij} x_j}{a^{(i-1)}_{ii}} \text{ para } i = n-1, n-2, \dots, 1 \quad (2.25)$$

En el algoritmo 3 se muestra el pseudocódigo de la eliminación gaussiana.

Algoritmo 3. Pseudocódigo de la eliminación gaussiana.

GaussianElimination (\mathbf{a} , \mathbf{b}):

Input: \mathbf{a} matrix of size $n * n$, \mathbf{b} vector of size n
Output: \mathbf{x} vector of size n

```

1: for k in 1 to n-1 do
2:   for i in k+1 to n do
3:     factor ←  $a_{i,k}/a_{k,k}$ 
4:     for j in k+1 to n do
5:        $a_{i,j} \leftarrow a_{i,j} - factor * a_{k,j}$ 
6:     end for
7:      $b_i \leftarrow b_i - factor * b_k$ 
8:   end for
10: end for
11:  $x_n \leftarrow b_n/a_{n,n}$ 
12: for i in n-1 to 1 do
13:   sum ←  $b_i$ 

```

```
14: for j in i+1 to n do
15:   sum ← sum +  $a_{i,j} * x_j$ 
16: end for
17:  $x_i \leftarrow sum/a_{i,i}$ 
18: end for
```

2.5 OpenCV

OpenCV (Open Source Computer Vision Library) es una librería de código abierto orientada a la visión computacional y aprendizaje máquina [30]. OpenCV se creó para proveer una infraestructura para aplicaciones de visión computacional y acelerar el uso de la percepción de las maquinas en productos comerciales. Inicio como un producto de licencia-BSD. OpenCV facilita a las empresas utilizar y modificar el código.

Esta librería incluye el conjunto clásico del arte de la visión computacional y los algoritmos de aprendizaje máquina. Estos algoritmos pueden ser utilizado para detectar y reconoce caras, identificar objetos, clasificar acciones humanas en videos, seguir los movimientos de las cámaras, seguir el movimiento de objetos, producir nubes 3D de puntos desde cámara estéreo entre otras más aplicaciones.

Esta librería es empleada por diversas compañías como Google, Yahoo!, Microsoft, Intel, IBM, Sony, Honda y Toyota. También empleada por gobiernos como el israelí para detectar intrusos, China en sus sistemas de monitoreo, en Turquía checando sus carreteras, en Japón para rápida detección de rostros. Y en general en aplicaciones de visión artificial en tiempo real.

Tiene interfaces en C++, Python, Java y Matlab, soportado en los sistemas operativos de Windows, Linux, Android y Mac Os. OpenCV está escrito nativamente

en C++ y tiene una interfaz con templates funcionando perfectamente con contenedores de la STL. Ver logo de la librería en la imagen 2.23.

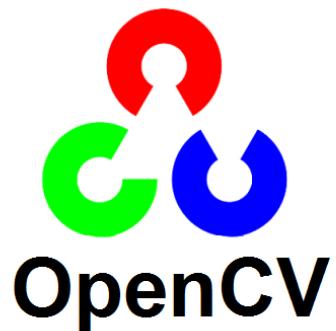


Figura 2.26. Logo de la librería OpenCV.

Capítulo 3 Metodología

En este capítulo analizaremos los aspectos generales a considerar en el diseño de un prototipo tipo AutoModelCar. El hardware y software empleados en la plataforma ro0botica. Así como, la metodología seguida en el diseño e implementación de los sistemas de visión artificial encargados de la detección de las líneas del carril e intersecciones de la pista de pruebas. El capítulo se divide en un total de siete secciones:

- Aspectos generales a considerar.
- Hardware y software.
- Diseño de la arquitectura.
- Adquisición de las imágenes.
- Preprocesamiento.
- Detección de las líneas del carril.
- Detección de intersecciones.

3.1 Aspectos generales a considerar

Los aspectos más importantes a tomar en cuenta en el desarrollo del prototipo, se basan en el reglamento de la competencia AutoModeCar del Torneo Mexicano de Robótica (TMR) en su edición 2019. A continuación, se enlistan las restricciones más importantes:

- La pista es una superficie de lona con fondo negro y líneas blancas, en ella se marcan dos carriles de 40 cm de ancho. La pista puede presentar reflexión de luz en superficie, debido al material que la constituye. El radio interno de la curvatura es de 1m, el radio de la línea punteada es de 1.4m y el externo es de 1.8m como se muestra en la figura 3.1.
- La iluminación de la pista es bajo condiciones de luz controlada (300 y 500 lx) y en un entorno cerrado. Las condiciones de iluminación pudieran variar en la competencia.
- El tamaño límite para una plataforma abiertas (de creación propia) debe ser de 50 cm de largo, 30 cm de ancho y 20 cm de alto.

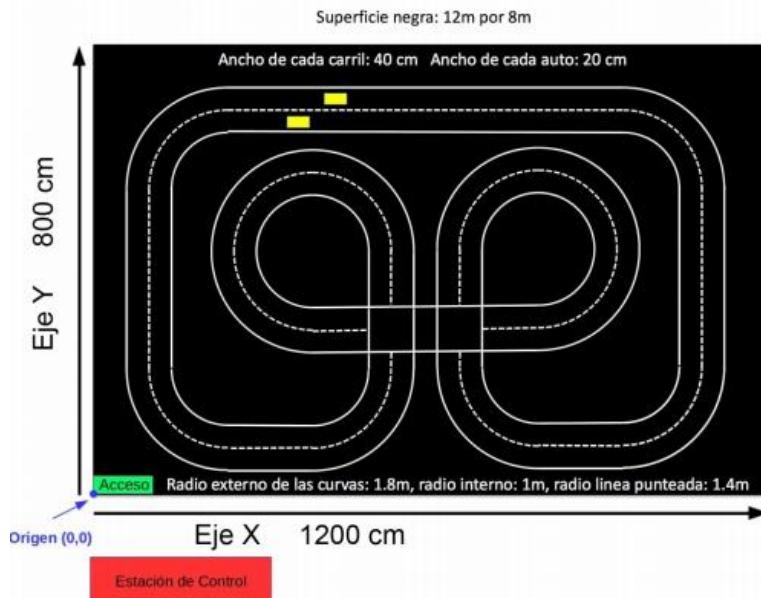


Figura 3.1. Pista de carreras.

Debido al gran tamaño de la pista de carreras de 8m x 12m del TMR. Se decidió armar se armó un tramo de 3 metros de pista con características similares a la pista

de la competencia ver figura 3.2, esto nos servirá para el desarrollo y pruebas de los algoritmos de visión computacional.

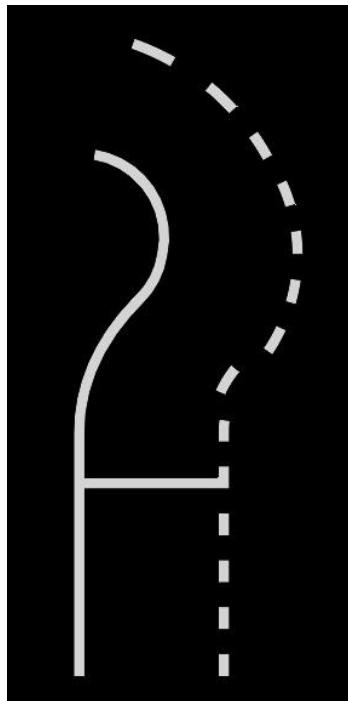


Figura 3.2. Tramo de pista para desarrollo y pruebas del sistema.

3.2 Hardware y software

Hardware

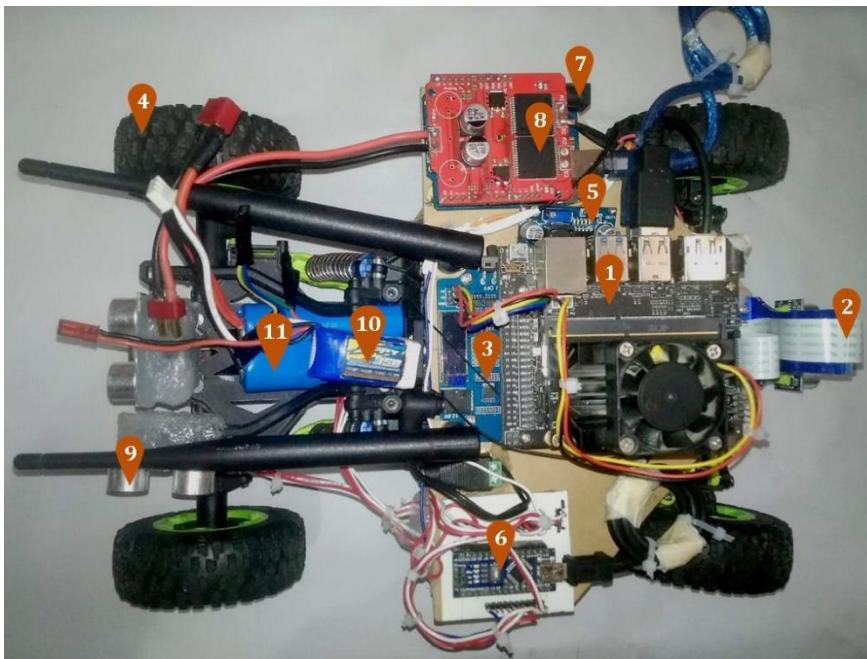
El procesamiento del sistema completo se debe llevar a bordo del automóvil a escala, por lo que se seleccionó una computadora embebida de tamaño reducido la NVIDIA Jetson Nano. Esta tarjeta corre una distribución personalizada de Linux basada en Ubuntu 18.04 provista por NVIDIA. La tarjeta esta específicamente diseñada para proyectos de inteligencia artificial, robótica, e internet de las cosas [12]. Cuenta con una interfaz (CUDA) entre la GPU y diversos lenguajes de programación como Python y C++. Así mismo la tarjeta dispone de una amplia documentación y extensa comunidad de desarrolladores que aportan soporte para el desarrollo de aplicaciones en esta plataforma.

La cámara seleccionada es el modelo Sony IMX219 de alta resolución por ser compatible con la interfaz MIPI CSI-2 DPHY de la tarjeta. Se selección un módulo

de expansión High Integrate JetBot, el cual provee a la tarjeta Jetson Nano de una alimentación de 5v a 3A, empleando 3 baterías 18650 o un cargador de 12v a 3A.

El automóvil eléctrico a escala seleccionado para montar nuestro sistema es el RC Car WLtoys 12428 4WD, con 24 cm de ancho, 36 cm de largo y 18 cm de alto. El coche cuenta con un motor DC cepillado de 7.2 V, el cual se conectó a un puente H el VNH2SP30 que soporta hasta 30A controlado por un Arduino Uno. El servomotor de 4.5kg-cm original del RC Car fue cambiado por un servomotor TowerPro MG995 de 15 kg-cm y se conectó a un Arduino Nano. El Arduino Nano también está conectado a tres sensores ultrasónicos HC-SR04 colocados en diferentes puntos del automóvil a escala. La integración completa del hardware se puede ver en la figura 3.3 con una vista superior y en la figura 3.4 con una vista frontal.

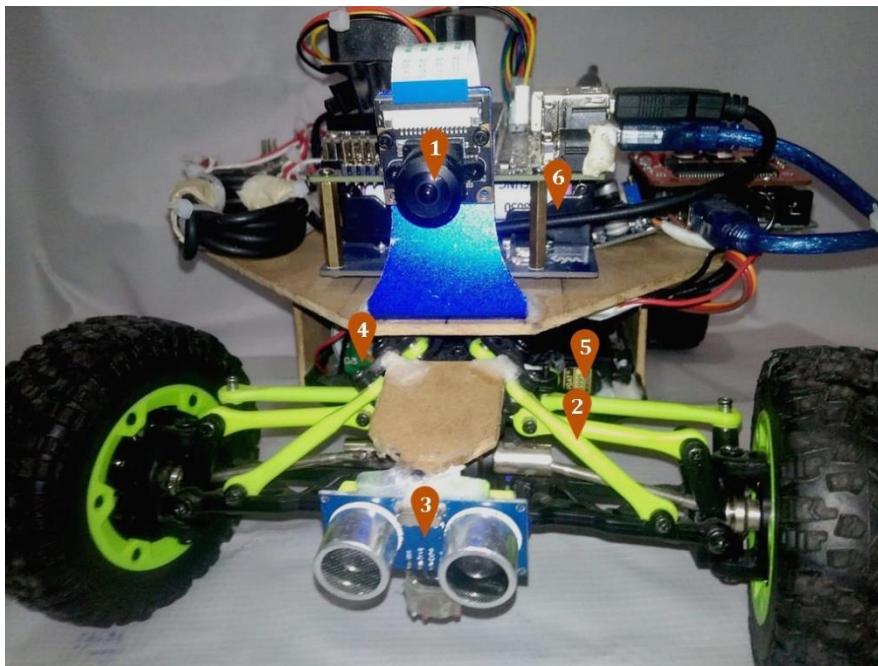
En la tabla 7 se puede observar la lista completa del hardware empleado y los costos por unidad de cada uno de los elementos utilizados.



VISTA SUPERIOR

1. NVIDIA Jetson Nano 128-Core GPU, 4-Core Cortex A-57 CPU, 4GB de RAM.
2. Cámara Sony IMX219, con una resolución de 3280 x 2464, con 8 megapíxeles.
3. High Integrate JetBot Expansion Board.
4. RC Car WLtoys 12428 4WD.
5. Módulo elevador de voltaje DC-DC XL6009E1.
6. Arduino Nano.
7. Arduino Uno.
8. Driver VNH2SP30 PWM Puente H 30A.
9. 3 sensores ultrasónicos HC-SR04.
10. Batería de 7.4 V a 350 mA.
11. Batería de 7.4 V a 1800 mA.

Figura 3.3. Vista superior de la integración.



VISTA FRONTAL

1. Cámara Sony IMX219, con una resolución de 3280 x 2464, con 8 megapíxeles.
2. RC Car WLtoys 12428 4WD.
3. 3 sensores ultrasónicos HC-SR04.
4. WLtoys 540 Cepillado Motor RC.
5. Servomotor TowerPro MG995 15Kg·cm.
6. 3 baterías 18650 Samsung 3.7 V a 2500 mA.

Figura 3.4. Vista frontal de la integración.

Tabla 7 Lista de hardware y costos.

Hardware	Cantidad de unidades empleas.	Costo por unidad. (peso mexicano- valuación enero 2020)
NVIDIA Jetson Nano 128-Core GPU, 4-Core Cortex A-57 CPU, 4GB de RAM.	1	\$2562
Cámara Sony IMX219, con una resolución de 3280 x 2464, con 8 megapíxeles.	1	\$670
High Integrate JetBot Expansion Board.	1	\$550

RC Car WLtoys 12428	1	\$1985
4WD.		
Módulo elevador de voltaje DC-DC XL6009E1.	1	\$45
Arduino Nano.	1	\$97
Arduino Uno.	1	\$150
Driver VNH2SP30 PWM	1	\$236
Puente H 30A.		
Sensor ultrasónico HC-SR04.	3	\$38
Servomotor TowerPro MG995 15Kg-cm.	1	\$104
Batería de 7.4 V a 1800 mA.	1	\$377
Batería de 7.4 V a 350 mA.	1	\$250
Batería 18650 Samsung 3.7 V a 2500 mA.	3	\$140
Cargador 12v-3A	1	\$300

Software

El sistema operativo a emplear es Ubuntu 18.04 provisto por la empresa NVIDIA. Sobre el cual se ha instalado el meta-sistema operativo ROS en su distro Melodic Morenia, este sistema funge como un integrador de todos los componentes del automóvil autónomo a escala. La librería de visión computacional a emplear es OpenCV en su versión 4.1, esta versión cuenta con la implementación de algoritmos de visión computacional tanto para CPU y como GPU , la aceleración por GPU se realiza por medio de la interfaz CUDA-OpenCV, se ha decidido no hacer uso de esta aceleración, principalmente para que los sistemas de detección de carril y cruce desarrollados se puedan ser reutilizar en otras plataformas embebidas que no cuenten con una GPU de NVIDIA, tal es el caso de la Raspberry Pi 4, la cual es una

plataforma más económica, pero no cuenta con una GPU de NVIDIA. En las implementaciones de los programas con ROS y OpenCV, existen dos principales lenguajes con los que se puede trabajar C++ y Python. C++ es un lenguaje compilado lo que permite una rápida ejecución del programa y alta eficiencia en el manejo de la memoria del programa haciéndolo ideal para implementar sistemas en tiempo real. Por otro lado tenemos a Python un lenguaje interpretado con una curva de aprendizaje bastante rápida, este lenguaje tiene una menor velocidad de ejecución en el CPU debido a la interpretación del leguaje, la velocidad de ejecución de los scripts de Python se puede mejorar, con la aceleración por GPU , pero esto limitaría el uso de estos programas únicamente a plataformas que cuente con una GPU NVIDIA además de integrar código en C, por estas razones las implementaciones de los programas se realizara en el lenguaje C++.

3.3 Diseño de la arquitectura

El paradigma robótico seleccionado para nuestro coche autónomo a escala es el paradigma jerárquico. Este paradigma tiene una relación jerárquica entre las primitivas del robot, primero la percepción de entorno, después la planificación de acciones a realizar y por último la ejecución de estas acciones.

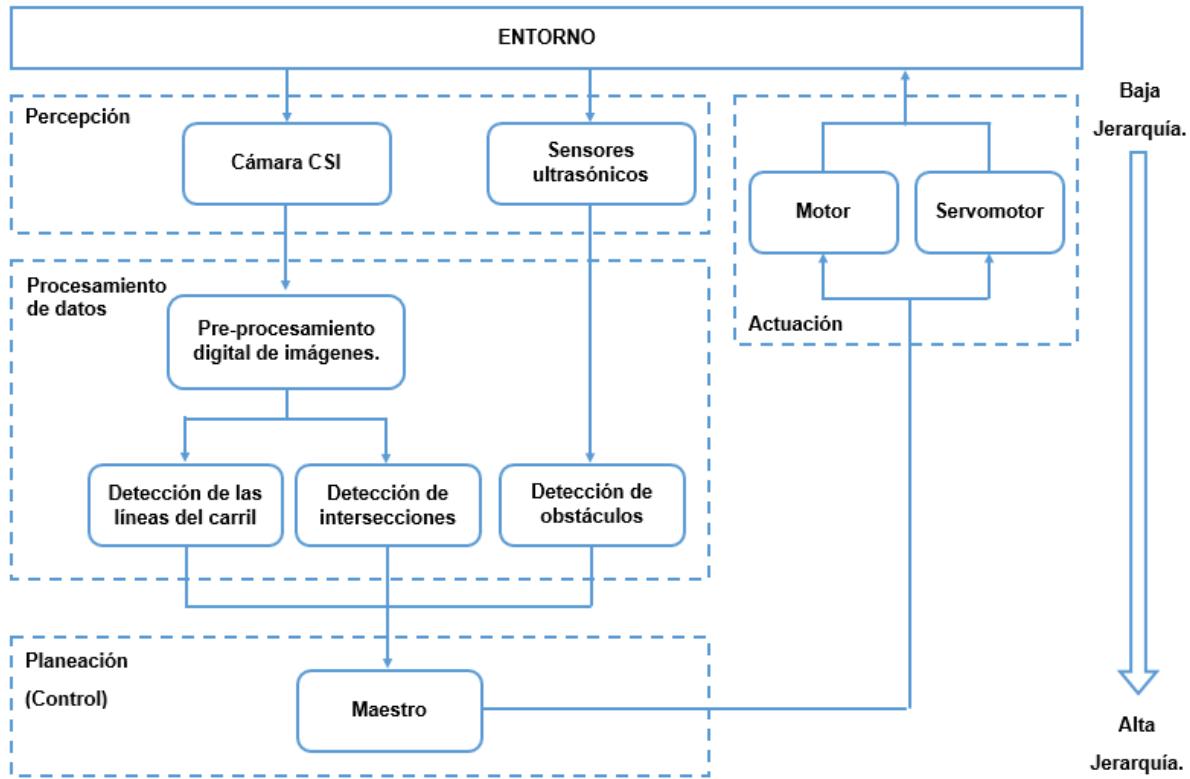


Figura 3.5. Arquitectura del sistema tipo AutoModelCar.

La arquitectura propuesta se puede ver en la figura 3.5. Esta arquitectura cuenta con las tres primitivas del paradigma jerárquico, más una capa intermedia entre las etapas de percepción y de planeación, la capa intermedia es la del procesamiento de datos la cual toma los datos crudos de la etapa de percepción y los transforma en información valiosa para la etapa de planificación. Las primitivas de más baja jerarquía son la percepción y la actuación, también son las etapas que tienen una interacción directa con el entorno de la pista de carreras. La etapa de planificación es la de mayor jerarquía, siendo también la etapa con un mayor nivel de abstracción del entorno.

Esta arquitectura está planteada para que se pueda trabajar de manera modular. Para lograrlo se emplea el concepto de los nodos de ROS y su interacción publicador-suscriptor por medio de tópicos y mensajes. El sistema se ha dividido en un total de 6 nodos. El primer nodo de encargar de adquirir la imagen desde la cámara, este nodo deberá correr en la Jetson Nano. El segundo nodo se deberá

ubicar en el Arduino Uno y se encargará del control de velocidad del Motor DC 540. El tercer nodo se ejecutará en un Arduino Nano este se encargará del control del servomotor y de la adquisición de datos de los sensores ultrasónicos, así como el procesamiento de estos datos. El cuarto nodo se centrará en la tarea de la detección de las líneas del carril y se ejecutará en la Jetson Nano. Un quinto nodo se empleará para la detección de intersecciones e igualmente correrá en la Jetson Nano. Un sexto nodo, el nodo maestro se encargará de la planificación y control del sistema en general, este nodo se encargará de planificar las acciones a tomar por los actuadores. Al emplear una arquitectura modular, el diseño y el desarrollo de cada uno de estos nodos se puede realizar de manera individual y luego integrar como la suma de todos los sistemas. A continuación, se trabajará con los nodos de adquisición de imágenes, detección de líneas del carril y la detección de los cruces.

3.3 Adquisición de las imágenes

Para lograr adquirir las imágenes provenientes de la cámara y enviarlas a la etapa de procesamiento de datos, se ha creado el nodo "jetson_camera" este nodo tiene la función de tomar una imagen desde el driver de la cámara IMX219-77, las dimensiones de la imagen nativa tomada por la cámara son de 3280 x 2464 pixeles. De estos pixeles solo se capturan 1440x2464 pixeles, que corresponde a la zona más representativa de la imagen, aun después de este recorte la imagen sigue teniendo una gran resolución porque se redimensiona la imagen a 288 x 492 píxeles como en la figura 3.6.



Figura 3.6. Imagen final para publicación.

Una vez obtenida una imagen como la figura 3.6 redimensionada para minimizar las operaciones de cómputo, se procede a publicarla en el tópico "/jetson_camera/raw" en un formato BGR. De manera que todos los nodos suscritos a este tópico pueden obtener una imagen de la cámara. Este nodo se encuentra en el paquete "/jetson_nano", el proceso general del nodo se muestra en la figura 3.7. La implementación completa del nodo se puede ver en el apéndice A jetson_camera.cpp.

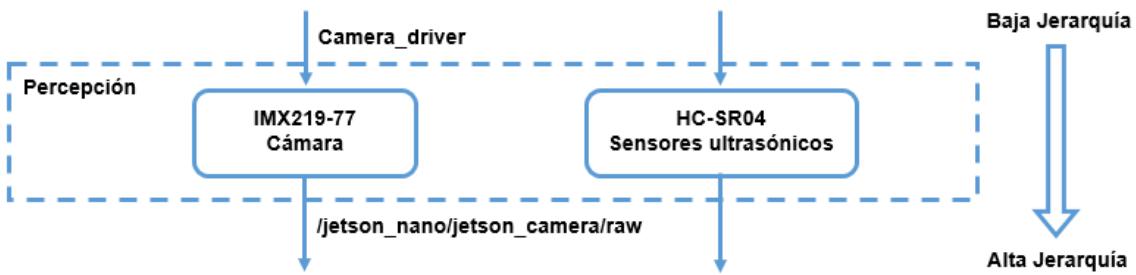


Figura 3.7. Adquisición de la imagen dentro de la arquitectura.

3.4 Preprocesamiento

Tanto el nodo que se encarga de la detección de las líneas del carril "lane_detection", como el nodo que encarga de la detección de intersecciones "line_crossing", se tienen que suscribir al tópico "/jetson_camera/raw" para obtener la imagen tomada por la cámara y proceder a aplicarle un preprocesamiento.

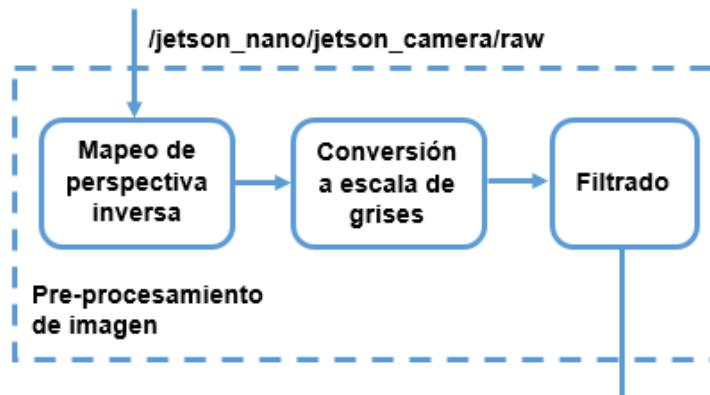


Figura 3.8. Preprocesamiento de la imagen.

El preprocessamiento está dividido en tres etapas (ver figura 3.8) que se describen a continuación:

3.4.1 Mapeo en perspectiva inversa.

De la imagen obtenida por suscripción al tópico "/jetson_camera/raw", se selecciona un área de interés, esta área debe contener la mayor cantidad de información sobre las líneas presentes en el carril de la pista, pero debe excluir información no representativa. Para poder delimitar esta área se emplea un método gráfico, el cual consiste en trazar un cuadrilátero por medio de cuatro puntos (156, 135), (336,135), (0,240) y (492,240) como en la figura 3.9.

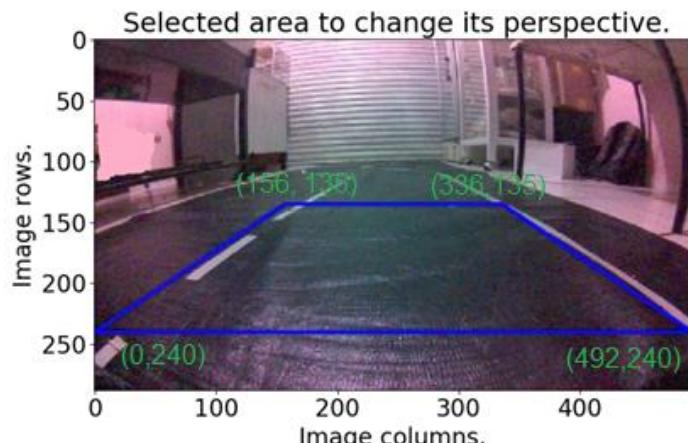


Figura 3.9. Área de interés.

A esta área de interés seleccionada se le aplica la técnica del mapeo en perspectiva inversa, teniendo como base de la perspectiva original los puntos de la figura 3.9. Estos puntos se trasladan a nueva imagen de las mismas dimensiones que la original, pero en una perspectiva inversa, los puntos de destino son (156, 0), (336,0), (156, 288) y (336,288) el resultado de esta operación se puede ver en la figura 3.10, donde la distorsión causada por el punto de fuga se ha eliminado.

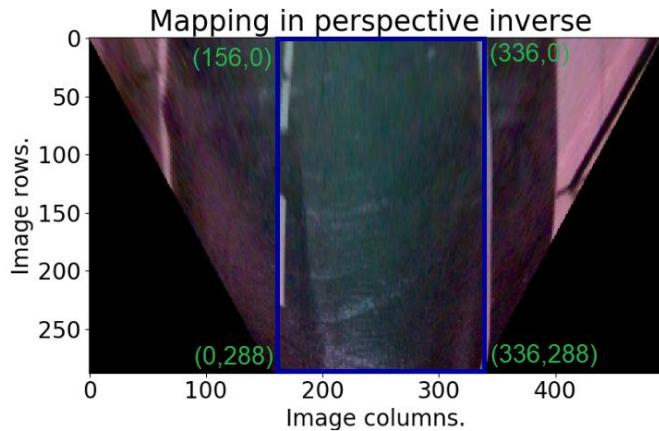


Figura 3.10. Mapeo en perspectiva inversa.

3.4.2 Conversión a escala de grises

Una vez eliminada la distorsión causada por el punto de fuga, se debe convertir la imagen de 3 canales de color (BGR) a una imagen de un solo canal de color en escala de grises ver figura 3.11, el motivo de realizar esta operación es principalmente reducir la dimensión de la matriz que representa la imagen de $R^3 \rightarrow R$. Reduciendo la cantidad de operaciones empleadas para la aplicación de filtros en la imagen.

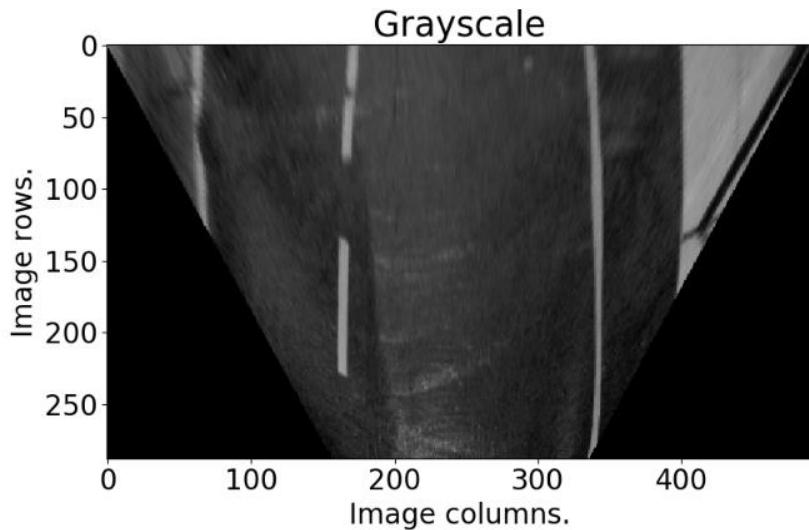


Figura 3.11. Conversión a escala de grises.

3.4.3 Filtrado

La aplicación de filtros a una imagen tiene principalmente dos finalidades, la remoción de ruido presente en la imagen y la segmentación de la imagen.

Remoción del ruido en la imagen.

Para poder eliminar el ruido causado por el brillo y rugosidad de la lona que constituye a la pista, se aplica un filtro mediano, el cual remplaza cada pixel de la imagen por la mediana de los píxeles vecinos. Al ser un filtro espacial requiere un kernel para realizar la operación de convolución en la imagen, el tamaño del núcleo influye considerablemente en la reducción del ruido en la imagen, por lo que un kernel pequeño no elimina suficiente ruido, pero un kernel demasiado grande podría causar una pérdida importante de la morfología original de la imagen, el tamaño del núcleo que se ha empleado es de 5, debido a que elimina bastante ruido en la imagen pero no hay gran pérdida morfológica ver figura 3.12.

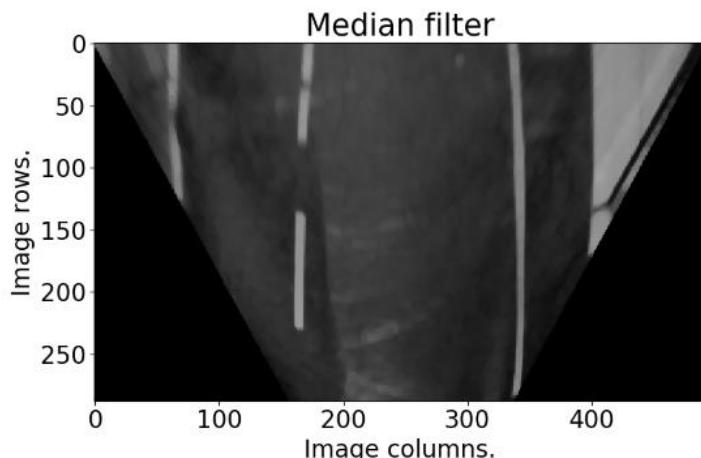


Figura 3.12. Aplicación del filtro mediana.

Segmentación de la imagen

La segmentación de imágenes es el proceso de dividir una imagen en partes o regiones. Esta división en partes se puede basar en las características de los píxeles de la imagen. Para este caso en particular se emplea una segmentación por umbral aplicando la ecuación 3.1 a cada uno de los píxeles en la imagen generando una nueva imagen $g(x, y)$ ver la figura 3.13, el valor de umbral es de 105 como

intensidad de umbral, donde el 0 corresponde al color negro de nula intensidad de color y 255 al blanco con máxima intensidad de color, el valor de umbral se obtuvo por medio de pruebas en diferentes condiciones de iluminación.

$$g[x, y] = \begin{cases} \text{valor de umbral}, & f[x, y] > \text{valor de umbral} \\ 0 & , f[x, y] \leq \text{valor de umbral} \end{cases} \quad (3.1)$$

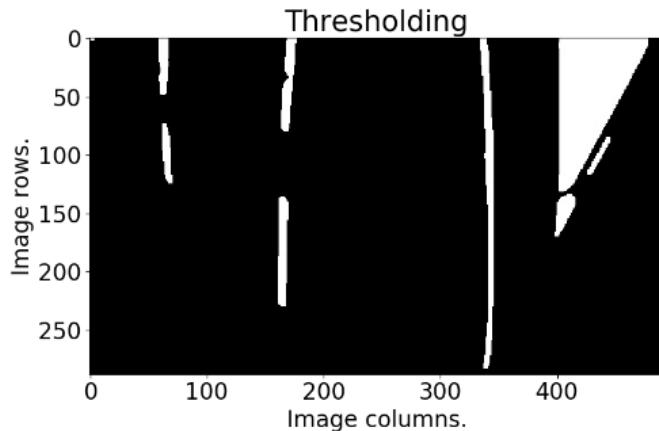


Figura 3.13. Segmentación por umbral.

Filtro de anchura blanca en dirección horizontal

De la figura 3.13 podemos observar un área blanca que no pertenece a la pista, esta área de gran anchura puede causar problemas en la búsqueda de los puntos pertenecientes a las líneas del carril o intersecciones, debido a las técnicas estadísticas que se emplean, siendo su desempeño afectado por este tipo de ruido. El problema causado se analizará más afondo después, por ahora lo importante es eliminar la mayor parte de estas áreas si afectar la morfología de las líneas del carril.

Para eliminar áreas de gran anchura, se implementó un filtro de anchura blanca el cual nos permite filtrar áreas menores a una determinada anchura, que presenten una intensidad de color mayor que 0. El filtro se aplica a cada fila de la imagen, convirtiendo la intensidad de color de áreas mayores que la anchura límite a una intensidad de color 0, el resultado de este filtro se puede ver en la figura 3.14.

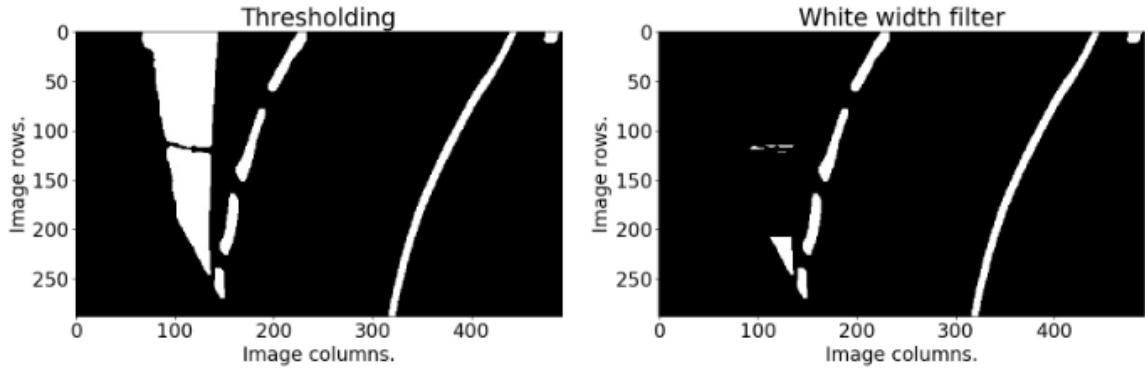


Figura 3.14. Efecto del filtro de anchura blanca.

La anchura máxima establecida por la anchura promedio del carril que es de 16 pixeles, obtenido de una muestra de 100 capturas en diversos segmentos de pista, más 8 pixeles de tolerancia por efectos de distorsión que pudieran presentar en la captura de la imagen, lo que nos da una anchura límite de 22 pixeles. El algoritmo de este filtro se muestra a continuación (ver algoritmo 4).

Algoritmo 4. Filtro de anchura en dirección horizontal.

width_filter (img, width_max):

Input:

img is a image of one color channel, **width_max** is the maximum width.

Output:

modified *img*.

```

1: white_left ← 0
2: white_right ← 0
3: rows ← img→rows
4: cols ← img→cols
5: for c in 0 to cols-1 do
6:   if img(rows/4, c)>0 or img(rows/2, c)>0 or img(rows * 3/4, c)>0 then
7:     if c < cols/2 then
8:       white_left ← white_left +1
9:     else then
10:      white_right ← white_right +1
11:    end if
12:  end if
13: end for
14: count_white ← 0
15: big_white ← false
16: last_point ← 0

```

```

17: now_point ← 0
18: black_to_white ← false
19: if white_right > white_left then
20:   for r in 0 to rows-1 do
21:     for c in 0 to cols-1 do
22:       if big_white = false then
23:         now_point ← img(r, c)
24:         if now_point > 0 then
25:           count_white ← count_white + 1
26:         end if
27:         if now_point > 0 and last_point = 0 then
28:           count_white ← 0
29:           black_to_white ← true
30:         else if now_point = 0 and last_point > 0 then
31:           black_to_white ← false
32:         end if
33:         if black_to_white = true and count_white >= width_max then
34:           c ← c-(width_max+1)
35:           big_white ← true
36:         end if
37:         last_point ← now_point
38:       else then
39:         img(r, c) ← 0
40:       end if
41:       big_white ← false
42:     end for
43   end for
44: else then
45:   for r in 0 to rows-1 do
46:     for c in cols-1 to 0 do
47:       if big_white = false then
48:         now_point ← img(r, c)
49:         if now_point > 0 then
50:           count_white ← count_white + 1
51:         end if
52:         if now_point > 0 and last_point = 0 then
53:           count_white ← 0
54:           black_to_white ← true
55:         else if now_point = 0 and last_point > 0 then
56:           black_to_white ← false
57:         end if
58:         if black_to_white = true and count_white >= width_max then
59:           c ← c+(width_max+1)
60:           big_white ← true
61:         end if
62:         last_point ← now_point

```

```

63:     else then
64:         img(r,c) ← 0
65:     end if
66:     big_white ← false
67: end for
68: end for
70: end if

```

3.5 Detección de las líneas del carril

El nodo que “percibe” y modela las líneas del carril es "lane_detection". Es un nodo que adquiere una imagen de la cámara suscriéndose al tópico "/jetson_camera/raw" y después se le aplica un preprocesamiento. En este punto aún no hemos detectado cuales son las líneas del carril y por lo tanto aún no conocemos un modelo matemático de las mismas, para lograr detectar los puntos que pertenece a cada línea del carril se han implementado una serie de técnicas y algoritmos de búsqueda que en conjunto pueden identificar que puntos en la imagen pertenecen a las líneas del carril. Una vez agrupados estos puntos, se procede a ajustar un polinomio de segundo grado a estos puntos y conseguir el modelo matemático de las líneas del carril, con este modelo podemos obtener la ubicación del coche respecto las líneas del carril, obteniendo la distancia del centro del carril al centro del coche, así como un ángulo de dirección del carril respecto al centro del coche. El proceso completo se puede observar en la figura 3.15.

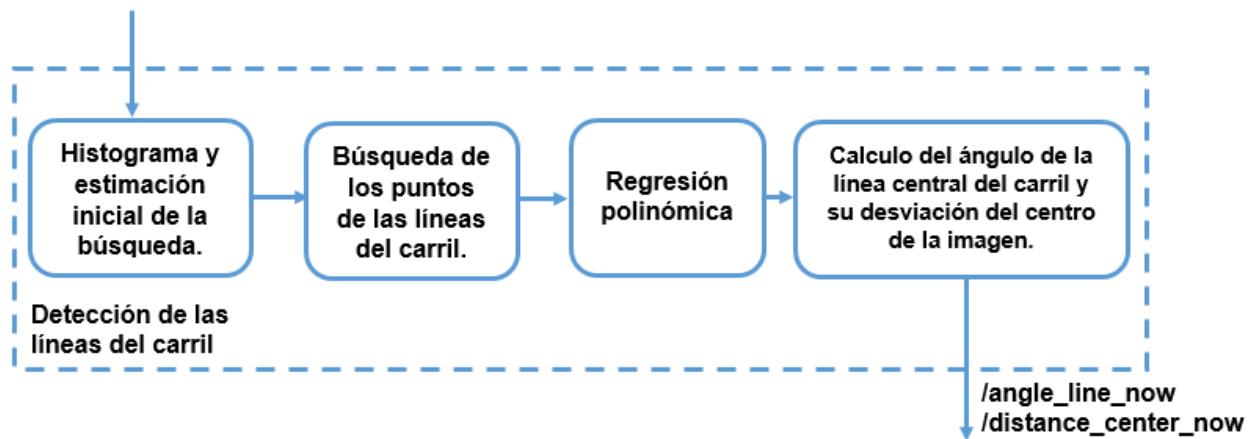


Figura 3.15. Proceso de la detección de las líneas del carril.

A continuación, analizaremos cada etapa del proceso de detección de las líneas del carril con más detalle.

3.5.1 Histograma y estimación inicial de la búsqueda

Para poder buscar los puntos pertenecientes a las líneas del carril, debemos tener un punto de partida de la búsqueda. La forma de estimar los puntos iniciales de la búsqueda del conjunto de puntos de cada línea, se emplea la técnica del histograma. El histograma que usamos analiza la distribución y concentración de puntos con una intensidad de color mayor que 0 en una imagen pre-procesada, en la dirección vertical de la imagen, sobre una determinada región de interés.

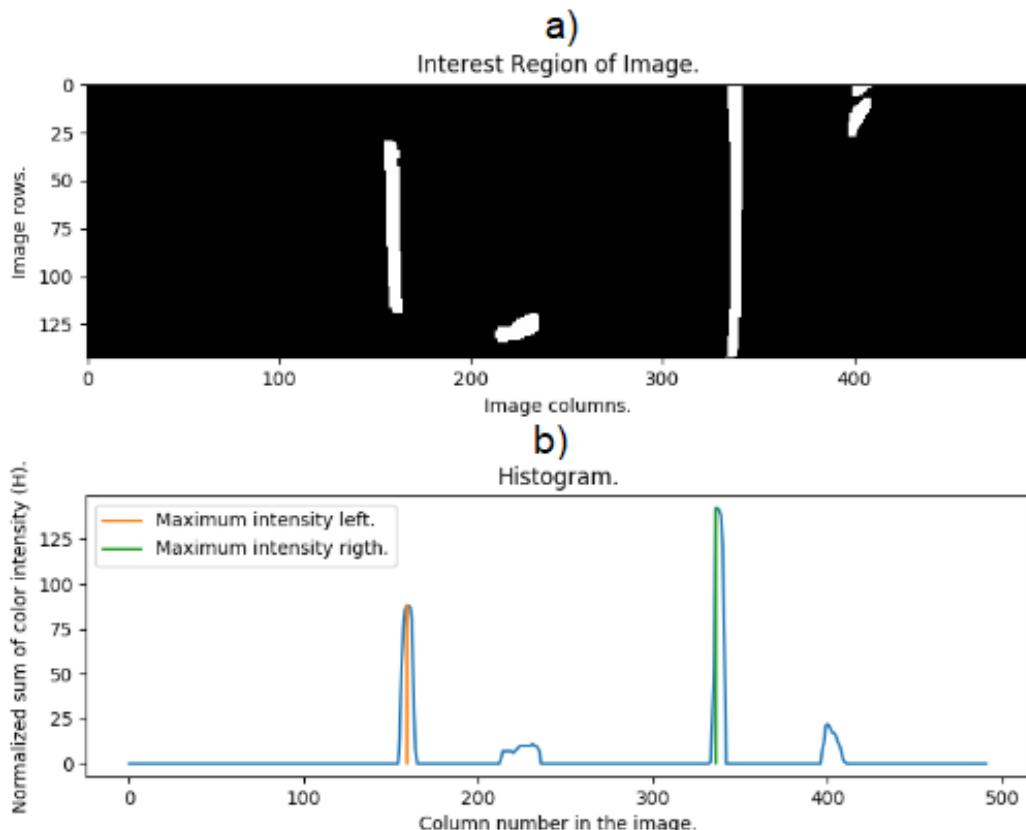


Figura 3.16. a) Región de interés, b) Histograma.

En la figura 3.16 a) se muestra la región de interés de 144 x 492 píxeles que corresponde a la mitad inferior de la imagen preprocesador de 288 x 492 píxeles. En la figura 3.16 b) podemos ver un histograma donde se representan la suma normalizada de las intensidades de color de todos los pixeles una columna de la

imagen para toda la región de interés, en esta figura 3.16 se observa que el máximo valor de la suma de intensidades normalizada corresponde a la columna donde se encuentra el inicio tanto de la línea derecha como el de la línea izquierda del carril, por lo que estas columnas sirven para estimar los puntos de inicio de la búsqueda de los pixeles de las líneas del carril.

Dada la información representada en el histograma, podemos percatarnos de un futuro fallo en la estimación de los puntos iniciales de búsqueda. Si existe una gran concentración de puntos con intensidades de color mayor que 0 que no pertenecen a las líneas del carril, se puede correr este valor máximo de la suma normalizada de las intensidades de color, desde las columnas de mayor concentración de puntos presentes en las líneas del carril hasta la concentración de puntos que no pertenece a las líneas del carril, obtenido una estimación errónea de los puntos iniciales de la búsqueda.

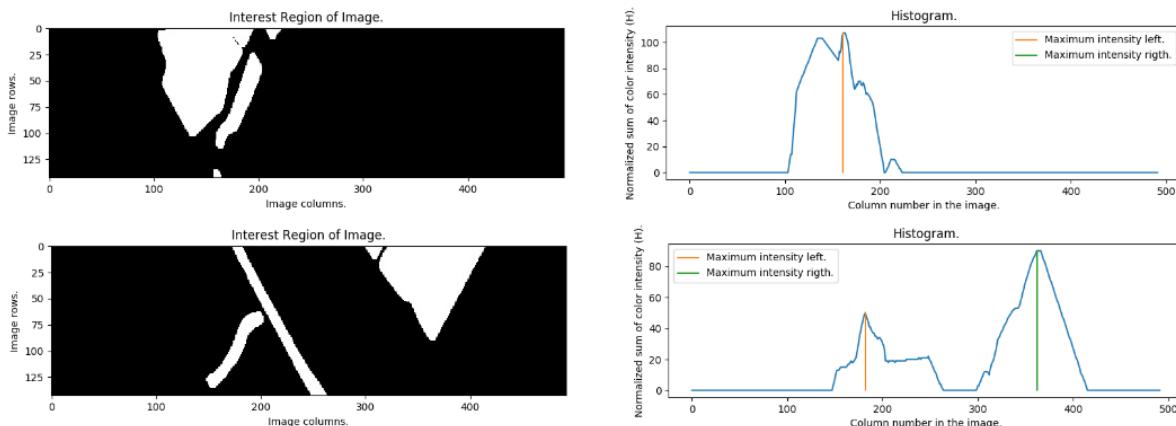


Figura 3.17. Errores en la estimación de los puntos iniciales de la búsqueda.

En figura 3.17, se muestran los dos errores más comunes en la estimación de los puntos iniciales de la búsqueda. El primero lo podemos ver en la parte superior de la figura, consiste en no detectar la línea del carril y solo detectar una región que no pertenece a la línea del carril como punto de partida de la búsqueda. El segundo error mostrado en parte inferior de la figura, consiste en detectar la línea del carril, pero también detectar la región que no pertenece a las líneas del carril como puntos de partida de la búsqueda. Estos errores se solucionan con el uso del filtro de anchura blanca ver figura 3.18.

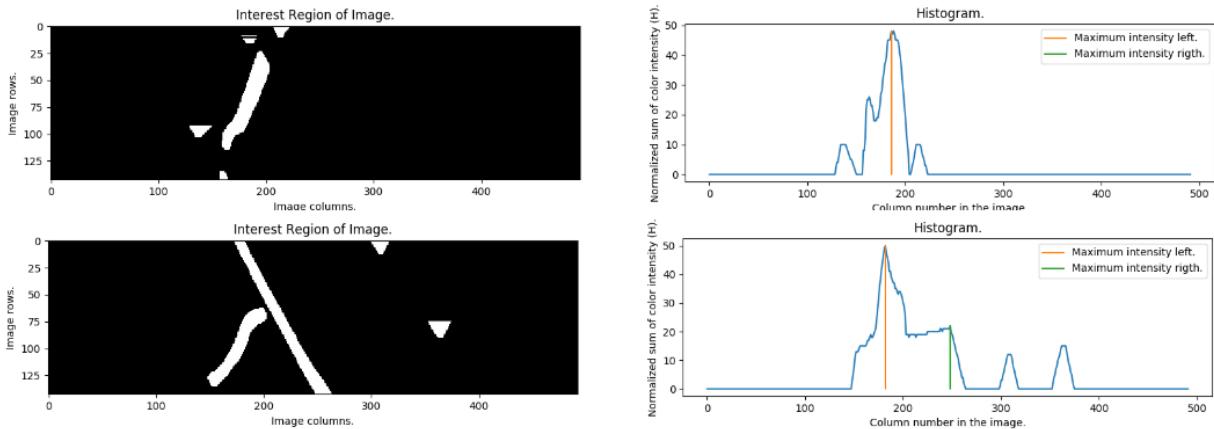


Figura 3.18. Solución de errores en la estimación de los puntos iniciales de la búsqueda.

3.5.2 Búsqueda de los puntos de las líneas del carril

Una vez se han estimado las columnas donde se encuentran los puntos iniciales de las líneas del carril, se debe emplear un algoritmo que nos permita agrupar todos los puntos que aporten información para lograr modelar matemáticamente las líneas del carril. El algoritmo seleccionado es el de la ventana deslizante.

Las ventanas de búsqueda, toman como punto medio las columnas donde se han estimado los puntos iniciales de la búsqueda, más un margen de 26 pixeles tanto a la derecha como a la izquierda, resultando en una ventana con una anchura de 56 pixeles y una altura de la ventana de búsqueda de 24 pixeles, por lo que para recorrer toda la imagen en una búsqueda vertical desde la fila 288 hasta la fila 0, se emplean 12 ventanas de búsqueda. En cada la ventana de búsqueda, se recorren la sección de las filas delimitada por la ventana, buscando los puntos con la máxima intensidad de color, estos puntos se van añadiendo a los datos que emplearemos para modelar las líneas del carril. Para actualizar el nuevo punto medio de la siguiente ventana de búsqueda, se ocupa el promedio del número de columna de todos los puntos encontrados en esa ventada con la mayor intensidad de color en cada fila, este punto medio solo se actualiza si la cantidad de puntos encontrados en esa ventana es mayor que 14. En la figura 3.19 podemos observar las ventanas de búsqueda agrupando a los pixeles que pertenecen a las líneas del carril. En el algoritmo 5 se muestra el procedimiento de la búsqueda de los puntos pertenecientes a las líneas del carril por medio de la ventana deslizante.

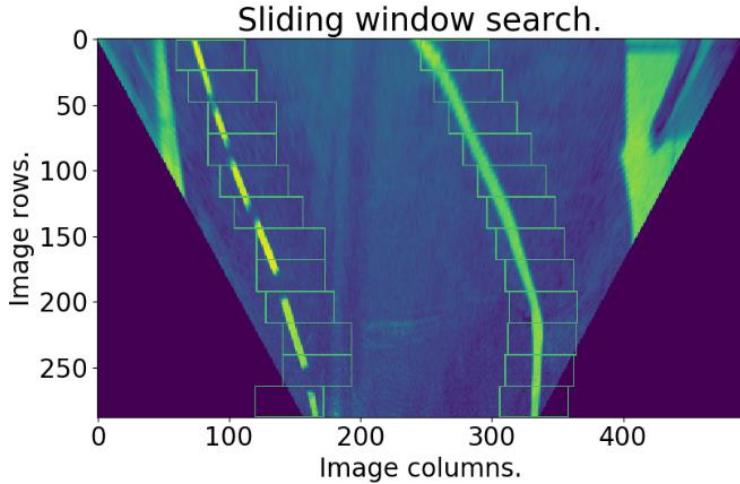


Figura 3.19. Búsqueda de las líneas del carril por medio de algoritmo de la ventana deslizante.

Algoritmo 5. Pseudocódigo de la búsqueda de los puntos de las líneas del carril por ventana deslizante.

locate_lane (img, locate_histogram_left, locate_histogram_right):

Input: *img* is a image of one color channel, **locate histogram left** and **locate histogram right** are estimated column number of the starting points of the lane lines.

Output:

left points is a points list of the left line and **right points** is a points list of the right line.

```

1: nwindows ← 12
2: margin ← 26
3: minpix← 14
4: rows ← img→rows
5: cols ← img→cols
6: isMaxLeft ← false
7: isMaxRight ← false
8: window_height ← rows/ nwindows
9: leftx_current ← locate_histogram_left
10: rightx_current ← locate_histogram_right
11: max_left ← 0
12: max_right ← 0
13: now_left_point ← 0
14: now_right_point ← 0
15: max_left_point ← (0,0)
16: max_right_point ← (0,0)
17: for window in 0 to nwindows-1 do
18:     mean_leftx ← 0
19:     for row in 0 to window_height-1 do
20:         for col in leftx_current - margin to leftx_current + margin do
21:             if pixel at (row, col) is white then
22:                 mean_leftx ← mean_leftx + col
23:                 count ← count + 1
24:             end if
25:         end for
26:         if count >= minpix then
27:             if mean_leftx > max_left then
28:                 max_left ← mean_leftx
29:                 max_left_point ← (row, col)
30:             end if
31:             if mean_leftx < min_left then
32:                 min_left ← mean_leftx
33:                 min_left_point ← (row, col)
34:             end if
35:         end if
36:         count ← 0
37:         mean_leftx ← 0
38:     end for
39:     leftx_current ← max_left_point
40:     if isMaxLeft = false or max_left > leftx_current then
41:         leftx_current ← max_left
42:         isMaxLeft ← true
43:     end if
44:     if isMaxRight = false or max_right < rightx_current then
45:         rightx_current ← max_right
46:         isMaxRight ← true
47:     end if
48: end for
49: left points ← list of points from (row, col) to (rightx_current, rightx_current)
50: right points ← list of points from (row, col) to (leftx_current, leftx_current)
51: end

```

```

18: mean_rightx ← 0
19: count_left ← 0
20: count_right ← 0
21: win_y_low ← rows-(window+1) *window_height
22: win_y_high ← rows-window*window_height
23: win_xleft_low ← leftx_current - margin
24: win_xleft_high ← leftx_current + margin
25: win_xright_low ← rightx_current - margin
26: win_xright_high ← rightx_current + margin
27: if win_xleft_low < 0 then
28:     win_xleft_low ← 0
29: end if
30: if win_xright_high > cols then
31:     win_xright_high ← cols
32: end if
33: if win_y_high > rows then
34:     win_y_high ← rows
35: end if
36: if win_y_low < 0 then
37:     win_y_low ← 0
38: end if
39: for r in win_y_low to win_y_high do
40:     max_left ← 0
41:     max_right ← 0
42:     isMaxLeft ← false
43:     isMaxRight ← false
44:     for cl in win_xleft_low to win_xleft_high do
45:         now_left_point ← img (r, cl)
46:         if now_left_point >0 and now_left_point >=max_left then
47:             max_left_point ← (r, cl)
48:             max_left ← now_left_point
49:             isMaxLeft ← true
50:         end if
51:     end for
52:     if isMaxLeft = true then
53:         left_points → push_back(max_left_point)
54:         mean_leftx ← mean_leftx + max_left_point →cl
55:         count_left ← count_left + 1
56:     end if
57:     for cr in win_xright_low to win_xright_high do
58:         now_right_point ← img(r, cr)
59:         if now_right_point >0 and now_right_point >= max_right then
60:             max_right_point ← (r, cr)
61:             max_right ← now_right_point
62:             isMaxRight ← true
63:         end if

```

```

64:    end for
65:    if isMaxRight = true then
66:        right_points → push_back(max_right_point)
67:        mean_rightx ← mean_rightx + max_right_point →cr
68:        count_right ← count_right + 1
69:    end if
70: end for
71: if count_left>=minpix then
72:    mean_leftx ← mean_leftx/ count_left
73:    leftx_current ← mean_leftx
74: end if
75: if count_right >=minpix then
76:    mean_rightx ← mean_rightx / count_right
77:    rightx_current ← mean_rightx
78: end if
79: end for

```

3.5.3 Regresión polinómica

Hasta este momento hemos logrado agrupar los puntos que pertenece a las líneas del carril, tanto de los puntos de la línea derecha como los de la línea izquierda. A partir de este grupo de datos podemos obtener un modelo matemático, el cual describa a cada línea del carril y emplearlos para posicionar al automóvil a escala dentro del carril de la pista en todo momento, aun si la pista presentase discontinuidades o irregularidades. El método seleccionado es la regresión polinómica, el cual nos permite ajustar un polinomio de segundo grado a un conjunto de datos agrupados, el modelo de la línea derecha y el de la línea izquierda se obtiene al solucionar un sistema de ecuaciones lineales de la forma $A = B^{-1}X$ como en la ecuación 3.2, donde los coeficientes del polinomio de segundo grado están dados por el vector $A = [a_0, a_1, a_2]$, m representa la cantidad de puntos grapados de una línea del carril , $\sum_{i=1}^m x_i$ es la sumatoria del número de columna de los puntos agrupados. $\sum_{i=1}^m x_i^2$ es la sumatoria del número de columna al cuadrado de estos puntos. $\sum_{i=1}^m x_i^3$ representa la sumatoria del cubo del número de columna de estos puntos. $\sum_{i=1}^m x_i^4$ es la sumatoria del número de columna a la

cuarta de los puntos encontrados y $\sum_{i=1}^m y_i$ representa la sumatoria del número de fila de los puntos agrupados pertenecientes a una línea del carril.

$$B = \begin{bmatrix} m & \sum_{i=1}^m x_i & \sum_{i=1}^m x_i^2 \\ \sum_{i=1}^m x_i & \sum_{i=1}^m x_i^2 & \sum_{i=1}^m x_i^3 \\ \sum_{i=1}^m x_i^2 & \sum_{i=1}^m x_i^3 & \sum_{i=1}^m x_i^4 \end{bmatrix}, A = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}, X = \begin{bmatrix} \sum_{i=1}^m y_i \\ \sum_{i=1}^m y_i * x_i \\ \sum_{i=1}^m y_i * x_i^2 \end{bmatrix} \quad (3.2)$$

Para resolver este sistema de ecuaciones lineales, podemos elegir cualquier método numérico que resuelva este tipo de sistemas, el método que se ha seleccionado es la eliminación gaussiana. La cual cuenta con dos fases principales: la eliminación de incógnitas hacia adelante obteniendo una matriz triangular superior y la segunda fase es la sustitución hacia atrás de las incógnitas del sistema, el pseudocódigo de este método numérico lo podemos ver en el algoritmo 3 (capítulo 2). En la figura 3.20, se observan los polinomios de segundo grado que se ajustan a los puntos a cada línea del carril.

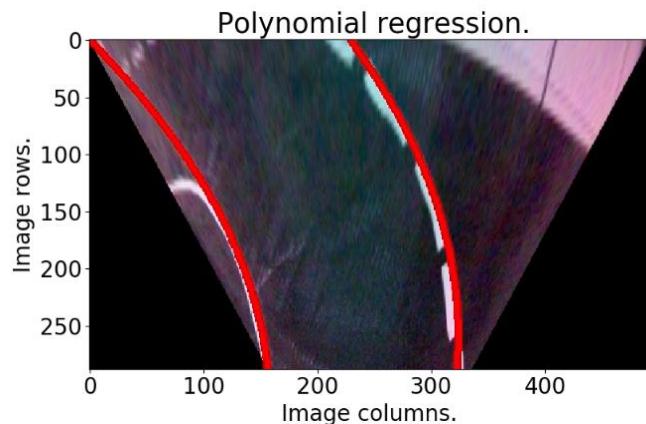


Figura 3.20. Polinomios de segundo grado ajustadas a los puntos de la línea derecha y los puntos de la línea izquierda del carril.

Para poder ajustar un nuevo polinomio de segundo grado al grupo de puntos de cada línea, el número de puntos debe ser mayor que 135, debido a que un número menor de puntos indica que no se ha encontrado un conjunto de puntos que aporte suficiente información para actualizar el modelo matemático. Los modelos que se obtienen describen una función cuya variable dependiente es una columna de la imagen y la variable independiente es una fila de la imagen, la función de la línea izquierda está dada por la ecuación 3.3 y la función de la línea derecha está dada por la ecuación 3.4.

$$columnL(row) = a_{l0} + a_{l1} * row + a_{l2} * row^2 \quad (3.3)$$

$$columnR(row) = a_{r0} + a_{r1} * row + a_{r2} * row^2 \quad (3.4)$$

Donde $A_l = [a_{l0}, a_{l1}, a_{l2}]$ es el vector solución del sistema de ecuaciones lineales obtenido de la línea izquierda del carril y $A_r = [a_{r0}, a_{r1}, a_{r2}]$ es el vector solución del sistema de ecuaciones lineales obtenido de la línea derecha del carril, $columnL$ y $columnR$ son columnas en la imagen y row es una fila en la imagen.

3.5.4 Calculo del ángulo de la línea central del carril y su desviación del centro de la imagen

Para lograr describir la posición del automóvil respecto al carril en que se encuentra conduciendo, se deben conocer principalmente dos parámetros, la desviación de la línea central del carril respecto al centro de la imagen donde se encuentra situada nuestra cámara y el ángulo en sentido antihorario que forma la línea central del carril con la horizontal de la imagen ver figura 3.21.

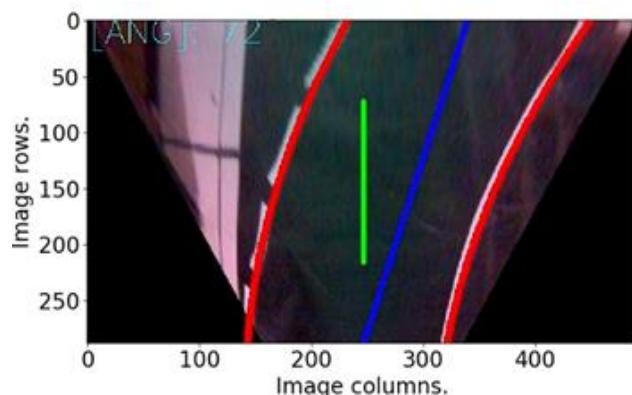


Figura 3.21. Línea central del carril.

En la figura 3.21 la línea verde representa el centro de la imagen y la ubicación de la cámara, las líneas rojas son las curvas ajustadas a los puntos de las líneas del carril, la línea azul representa la línea central del carril, esta línea nos indica la dirección del carril con respecto al centro del coche, esta dirección se determina por el ángulo formado por la línea central del carril y la horizontal de la imagen. Para calcular el ángulo en grados de la línea central con la horizontal de la imagen y su desviación del centro de la imagen se emplea el algoritmo 6.

Algoritmo 6. Cálculo del ángulo de la línea central del carril y su desviación del centro de la imagen.

car_position_in_lane (img, A_l , A_r):

Input:

img is a image of one color channel, \mathbf{A}_l is a vector of the coefficients of the left polynomial, \mathbf{A}_r is a vector of the coefficients of the right polynomial.

Output:

angle is an angle (degrees) of the center line of the lane, distance_center is the deviation of the center line of the lane from the center of the image.

```

1: rows ← img→rows
2: cols ← img→cols
3: center_cam ← cols/2
4: columnR ←  $a_{r0} + a_{r1} * (rows/2) + a_{r2} * (rows/2)^2$ 
5: columnL ←  $a_{l0} + a_{l1} * (rows/2) + a_{l2} * (rows/2)^2$ 
6: center_lines ← (columnR + columnL)/2
7: distance_center ← center_cam - center_lines
8: if distance_center = 0 then
9:   angle ← 90
10: else then
11:   angle_to_mid_radian ←  $\tan^{-1}((rows/2 - rows)/(center_lines - center_cam))$ 
12:   angle ← angle_to_mid_radian * 57.295779
13:   if angle < 0 and angle > -90 then
14:     angle ← (0-1) * (angle)
15:   else if angle > 0 and angle < 90 then
16:     angle ← 180 - angle
17:   end if
18: end if

```

Una vez hemos obtenido el ángulo y la desviación en pixeles al centro de la imagen, de la línea central del carril, se deben publicar estos dos parámetros para ser

empleados por otros nodos. El ángulo de la línea central del carril se publica en el tópico "/angle_line_now". La desviación al centro de la imagen de la line central del carril se publica en el tópico "/distance_center_line". La implementación completa del nodo "lane_detection" se puede encontrar en el apéndice B.

3.6 Detección de intersecciones

El nodo que se encarga de detectar los cruces en el camino es "crossing_detection", este nodo adquiere una imagen de la cámara suscribiéndose al tópico "/jetson_camera/raw" y después pasa por un preprocesamiento. Para lograr detectar las intersecciones en la pista a la por medio de una imagen preprocesada, se le aplican un proceso muy similar al empleado en el nodo "lane_detection" para detectar las líneas del carril de la pista ver figura 3.22.

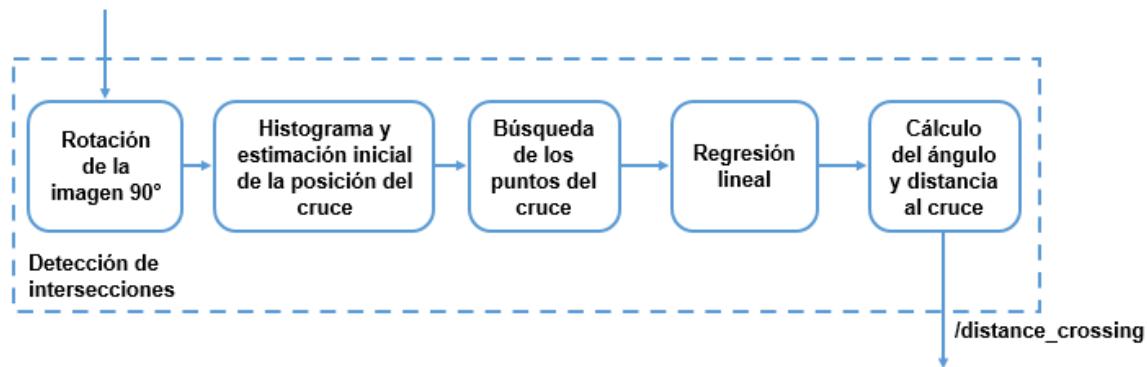


Figura 3.22. Proceso de la detección del cruce.

Como se puede ver en la imagen 3.22 el proceso de detección del cruce cuenta con una etapa de rotación de la imagen debido al que el cruce se encuentra en posición horizontal y para aplicar el proceso de detección de líneas ya desarrollado, se debe colocar el cruce en posición vertical (ver figura 3.23). A continuación se analizan las adaptaciones a cada etapa del proceso de detección de líneas del carril para poder detectar cruces.

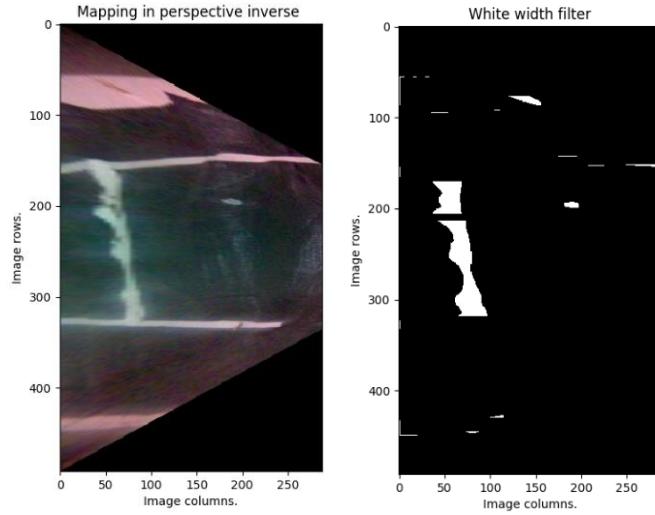


Figura 3.23. Rotación de la imagen preprocesada.

Histograma y estimación inicial de la posición del cruce

El histograma que hemos empleado representa la suma normalizada de las intensidades de color de todos los píxeles en una columna, una determina región de interés de la imagen preprocesada del carril ver figura 3.24. En este nodo buscamos una única columna con el máximo valor de intensidad, la cual indica la estimación de la búsqueda de los puntos pertenecientes al cruce.

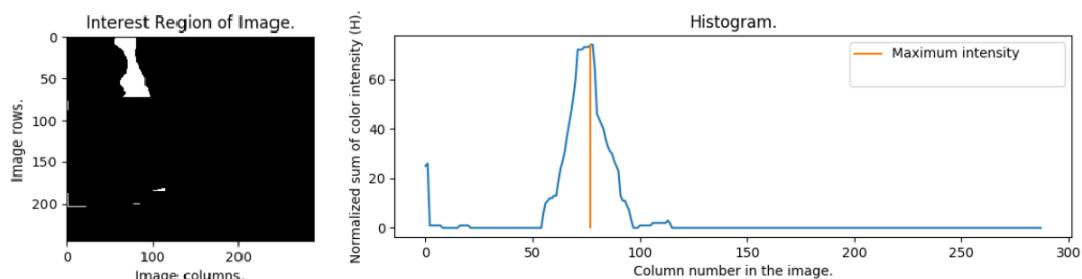


Figura 3.24. Región de interés e histograma.

Búsqueda de los puntos del cruce

Una vez se ha estimado la columna donde se encuentran el punto inicial del cruce, se debe emplear el algoritmo de la ventana deslizante, para agrupar todos los puntos que aporten información para el modelo matemáticamente del cruce. El número de ventanas a emplear es 12, con un tamaño de la ventana de búsqueda de 40 píxeles de anchura y 41 píxeles de altura. El número mínimo de píxeles para

actualizar el punto medio de la ventana es 14 píxeles. En la figura 3.25 se muestra la búsqueda por ventana deslizante de los puntos que pertenecen al cruce.

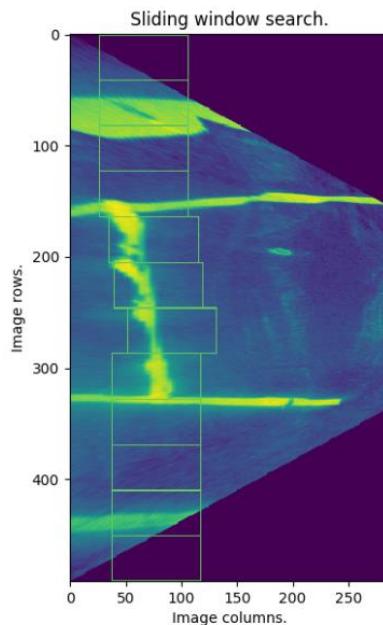


Figura 3.25. Búsqueda por ventana deslizante de los puntos que pertenecen al cruce.

Regresión lineal

A partir los puntos encontrados podemos obtener un modelo que describa la línea del cruce. Para conseguir este modelo se emplea una regresión lineal sobre los datos encontrados, por medio de un sistema de ecuaciones lineales de la forma $A = B^{-1}X$ como en la ecuación 3.5, donde los coeficientes del modelo lineal están dados por el vector $A = [a_0, a_1]$, m representa la cantidad de puntos grafados de la línea del cruce. $\sum_{i=1}^m x_i$ es la sumatoria del número de columna de los puntos agrupados de la línea del cruce. $\sum_{i=1}^m x_i^2$ es la sumatoria del número de columna al cuadrado de los puntos encontrados y $\sum_{i=1}^m y_i$ representa la sumatoria del número de fila de todos los puntos agrupados de la línea del cruce. Para resolver este sistema de ecuaciones lineales se hace uso de la eliminación gaussiana.

$$B = \begin{bmatrix} m & \sum_{i=1}^m x_i \\ \sum_{i=1}^m x_i & \sum_{i=1}^m x_i^2 \end{bmatrix}, A = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}, X = \begin{bmatrix} \sum_{i=1}^m y_i \\ \sum_{i=1}^m y_i * x_i \end{bmatrix} \quad (3.5)$$

El modelo encontrado describe una función cuya variable dependiente es una columna de la imagen y la variable independiente es una fila de la imagen, la función de la línea del cruce está dada por la ecuación 3.6.

$$\text{column_c}(row) = a_0 + a_1 * row \quad (3.6)$$

Donde $A = [a_0, a_1]$ es el vector solución del sistema de ecuaciones lineales obtenido de los puntos de la línea del cruce, column_c es una columna en la imagen y row es una fila en la imagen. El número mínimo de pixeles para actualizar el modelo es de 100.

Cálculo del ángulo y distancia al cruce

Para poder saber si la línea encontrada es un cruce y su distancia al coche, se debe determinar el ángulo entre la línea del cruce respecto a la horizontal de la imagen, este ángulo debe ser de 90° con un margen de $\pm 15^\circ$, la distancia al coche se toma desde la columna $\text{column_c}\left(\frac{\text{filas de la imagen}}{2}\right)$ a la última columna de la imagen.

Para calcular estos parámetros empleamos el algoritmo 7.

Algoritmo 7. Cálculo del ángulo y distancia al cruce.

crossing_line (img, \mathbf{A}_c)

Input:

img is a image of one color channel, \mathbf{A}_c is a vector of the coefficients of the crossing line polynomial

Output:

angle is an angle (degrees) of the crossing line , *distance_to_crossing* is the distance from the car to the crossing line.

1: rows \leftarrow img \rightarrow rows

```

2: cols ← img→cols
3: column1 ←  $a_{c0} + a_{c1} * (rows/2)$ 
4: column2 ←  $a_{c0} + a_{c1} * (rows * 3/4)$ 
5: distance_to_crossing ← cols - column1
6: angle_to_mid_radian ←  $\tan^{-1}((rows/2 - rows * 3/4)/(column1 - column2))$ 
7: angle ← angle_to_mid_radian * 57.295779
8: if angle < 0 and angle > -90 then
9:   angle ← (0-1) * (angle)
10: else if angle > 0 and angle < 90 then
11:   angle ← 180 - angle
12: else then
13:   angle ← 0
14: end if

```

En la figura 3.26 se puede observar de una manera gráfica la distancia dada en columnas que se obtiene desde la línea del cruce hasta la última columna de la imagen donde se ubica la parte frontal de coche. Si el ángulo de la línea del cruce estuviera fuera del rango de $90^\circ \pm 15^\circ$ la distancia del coche al cruce se establecería en cero. Una vez se ha validado la distancia del cruce al carro, esta se publica al tópico “/distance_crossing” para ser empleada por algún otro nodo. La implementación completa del nodo “crossing_detection” se puede ver en el apéndice C crossing_detection.cpp.

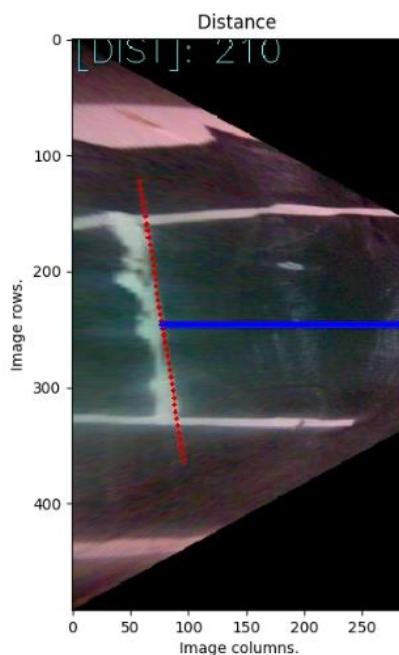


Figura 3.26. Distancia en columnas desde la línea del cruce hasta el coche.

Capítulo 4 Resultados experimentales

En este capítulo se analizan los resultados experimentales obtenidos de la implementación del subsistema de visión artificial para la percepción de las líneas del carril e intersecciones en la pista de pruebas con características similares a la pista de la competencia AutoModelCar, el capítulo cuatro se divide en cinco secciones:

- Sección recta.
- Sección curva A y B.
- Sección intersección.
- Comparativa del desempeño de los modelos de las líneas del carril e intersecciones.
- Frecuencias de procesamiento de los nodos del subsistema de visión artificial.

Para evaluar los sistemas de visión artificial obtenidos se ha dividido la pista de pruebas en cuatro secciones de pruebas ver figura 4.1, estas secciones presentan las características más relevantes de la pista de carreras en categoría AutoModelCar del TMR. Por cada sección de pruebas se tomaron 6 imágenes muestra, por cada una de estas imágenes obtenemos estadísticas descriptivas que ayudan a visualizar los resultados del algoritmo de búsqueda de los puntos pertenecientes a las líneas del carril y las intersecciones, por cada sección de pruebas. Para evaluar los modelos obtenidos que representar las líneas del carril y las intersecciones analizamos el error generado en cada sección de la pista de pruebas.

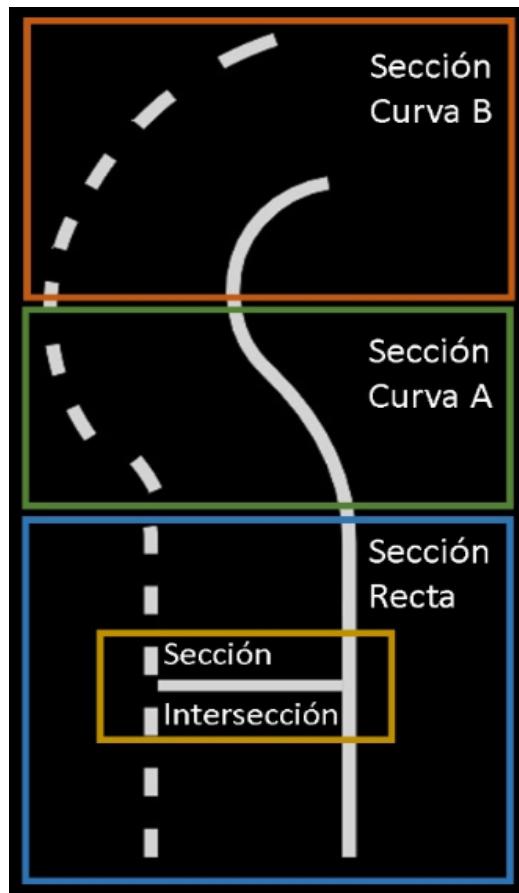


Figura 4.1. Secciones de la pista de pruebas.

4.1 Sección recta

En la figura 4.2 tenemos seis diferentes imágenes capturas a lo largo de la sección recta de la pista, las cuales ya se han pasado por las etapas de preprocesado. A estas imágenes ya preprocesadas, se les aplica el algoritmo de la búsqueda por ventana deslizante para agrupar los puntos que pertenecen a cada línea del carril en el tramo recto.

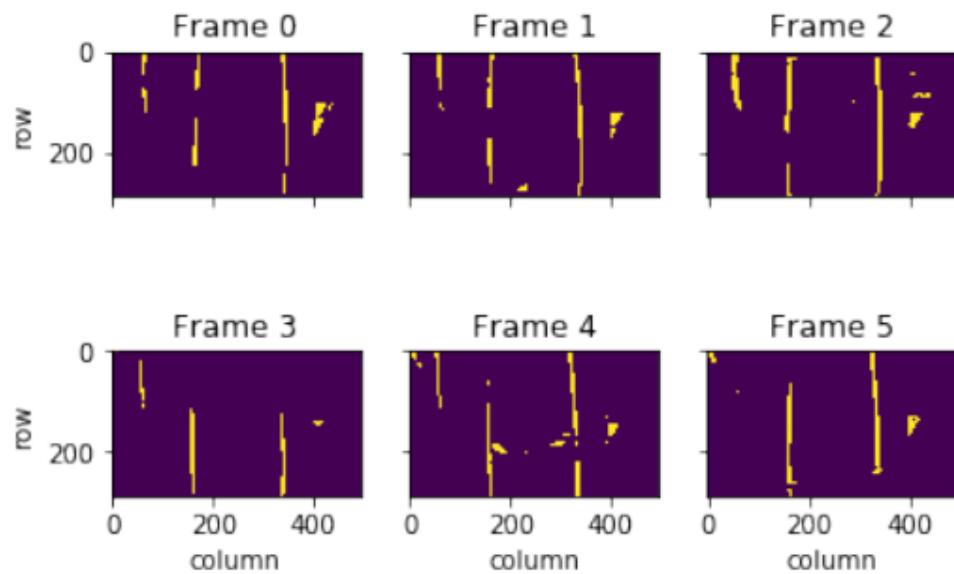


Figura 4.2. Imágenes preprocesadas de la sección recta.

De las búsquedas verticales realizadas por el algoritmo desde la fila 0 hasta la fila 288 de cada imagen muestra que se tomó en la sección recta, obtenemos un conjunto de puntos por muestra pertenecientes a las líneas izquierdas y derechas, en la figura 4.3 se visualizan las posiciones de los puntos agrupados para las líneas izquierdas y en la figura 4.4 se visualizan las posiciones de los puntos agrupados para las líneas derechas.

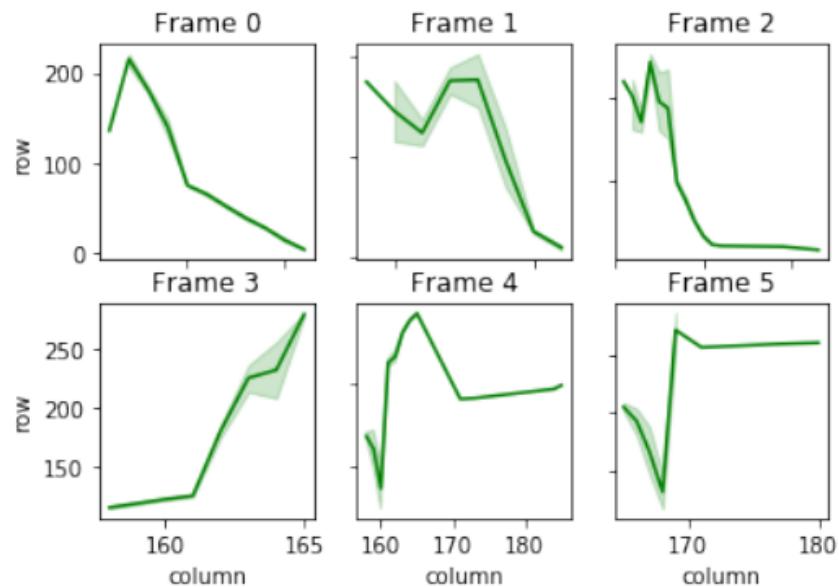


Figura 4.3. Posiciones en la imagen muestra de los puntos pertenecientes a las líneas izquierdas en la sección recta.

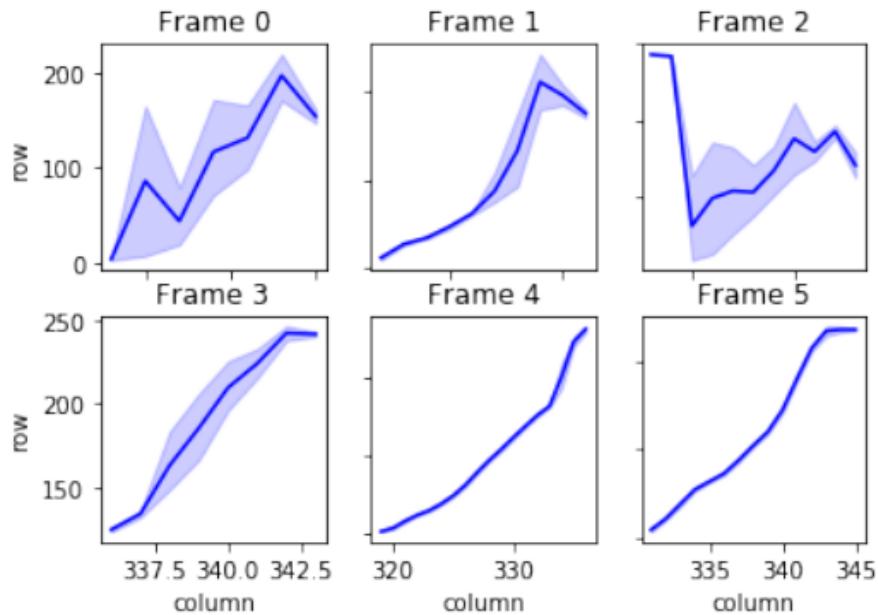


Figura 4.4. Posiciones en la imagen muestra de los puntos pertenecientes a las líneas derechas en la sección recta.

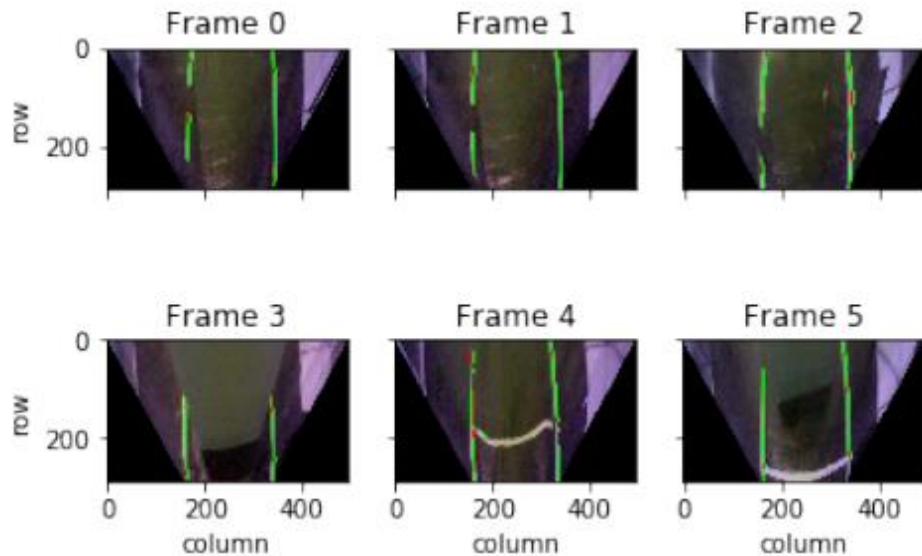


Figura 4.5. Conjuntos de puntos agrupados que pertenecen a las líneas del carril de la sección recta.

En la figura 4.5 se observan los conjuntos de puntos agrupados sobre las líneas izquierdas y derechas del carril. De dichos conjuntos de datos podemos obteneremos las siguientes estadísticas descriptivas, dispersiones estándares y medianas, tomando como variable de estudio la posición de columna y no de la fila, debido a que los modelos obtenidos muestran la predicción en columnas y no en filas, las estadísticas se pueden ver en la tabla 8.

Tabla 8. Desviaciones estándares y medianas de las columnas de los puntos agrupados alrededor de las líneas del carril en el tramo recto.

Número de frame.	Mediana de la línea izquierda (posición en columna).	Desviación estándar de la línea izquierda. (posición en columna)	Mediana de la línea derecha (en columna).	Desviación estándar de la línea derecha. (posición en columna)
0	170.13450	2.76560	343.87142	1.52522

1	162.92079	1.51352	338.38811	2.84808
2	166.30697	3.35755	345.315018	2.39880
3	162.44970	1.23992	340.042424	1.82690
4	162.11009	6.51228	329.692622	4.88806
5	166.96226	1.82707	338.107883	3.78969

De la tabla 8 tenemos las columnas promedio de todos los puntos agrupados alrededor de las líneas izquierdas y derecha, también se muestran las desviaciones estándares de la pasión de columna promedio que presentan las agrupaciones de puntos por frame. Para realizar una observación visual de la distribución de los puntos agrupados alrededor de las líneas carril en recta de cada frame muestra, se tienen las figuras 4.4 y 4.5, en estas figuras presenta las distribuciones de puntos agrupados en función de su posición de columna en la imagen muestra, al ser las líneas de la sección recta, las distribuciones presentadas no son muy dispersas, encontrándose los puntos de las líneas del carril en rangos de columnas bastante reducidos.

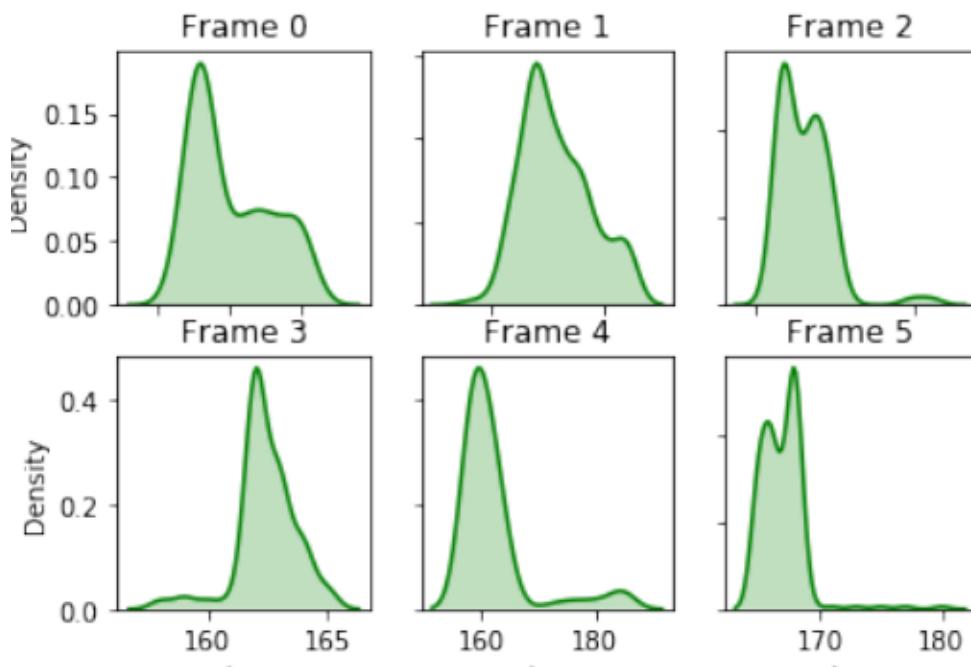


Figura 4.6. Distribución de puntos agrupados de las líneas izquierdas en el tramo recto.

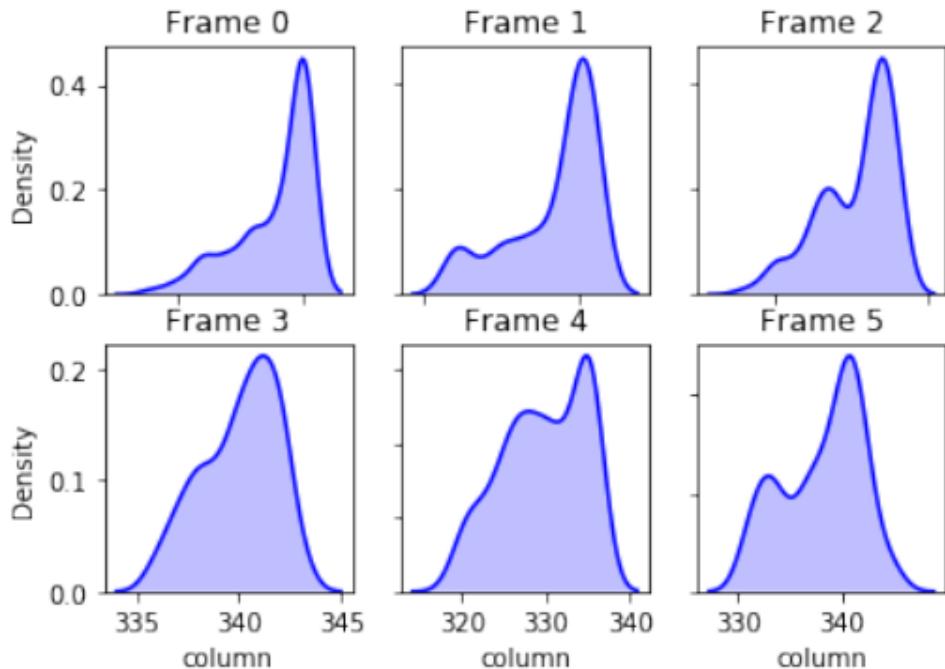


Figura 4.7. Distribución de puntos agrupados de las líneas derechas en el tramo recto.

A partir de los puntos agrupados se obtienen los modelos de las líneas izquierdas y derechas del carril, por medio de regresiones polinomiales, las cuales ajustan polinomios de segundo grado sobre conjunto de puntos agrupados a cada línea del carril. Para evaluar los modelos de regresión obtenidos se emplean dos métricas, la métrica del error absoluto medio (MAE) ecuación 4.1, el cual es una métrica robusta que no varía por valores extremos encontrados en los datos, esta métrica se puede interpretar como unidades de la variable objetivo la cual es la columna del punto perteneciente a la línea del carril; la segunda métrica es la raíz del error cuadrático medio (RMSE) ecuación 4.2, el resultado se puede medir en las mismas unidades de la variable objetivo, pero dando una mayor penalización a errores grandes.

$$\frac{1}{n} \sum_{i=0}^n |y_i - \hat{y}_i| \quad (4.1)$$

$$\sqrt{\frac{1}{n} \sum_{i=0}^n (y_i - \hat{y}_i)^2} \quad (4.2)$$

Donde n es el número de datos agrupados por línea de carril, y_i es la variable objetivo dada en la posición de columna del punto agrupado y \hat{y}_i es el pronóstico de la posición de columna del punto agrupado. En la tabla 9 se muestran los errores RMSE y MAE de los modelos que describen las líneas izquierdas y derechas del carril para cada frame muestra tomado en la sección recta.

Tabla 9. Errores de los modelos de las líneas del carril en el tramo recto.

Frame	MAE_LEFT (columnas)	RMSE_LEFT (columnas)	MAE_RIGHT (columnas)	RMSE_RIGHT (columnas)
0	0.608187	0.787327	0.496429	0.709628
1	0.519802	0.727807	0.583916	0.777750
2	0.916279	1.606672	0.886447	1.228478
3	0.763314	1.095985	0.575758	0.827556
4	3.605505	6.277234	0.512295	0.721451
5	0.900943	1.658312	0.535270	0.817343

En la figura 4.8 se muestra el resultado de las líneas generadas a partir de la aplicación de los modelos obtenido, consiguiendo una posición de columna por cada fila de la imagen muestra, con estas posiciones columna y fila es posible graficar las líneas del carril.

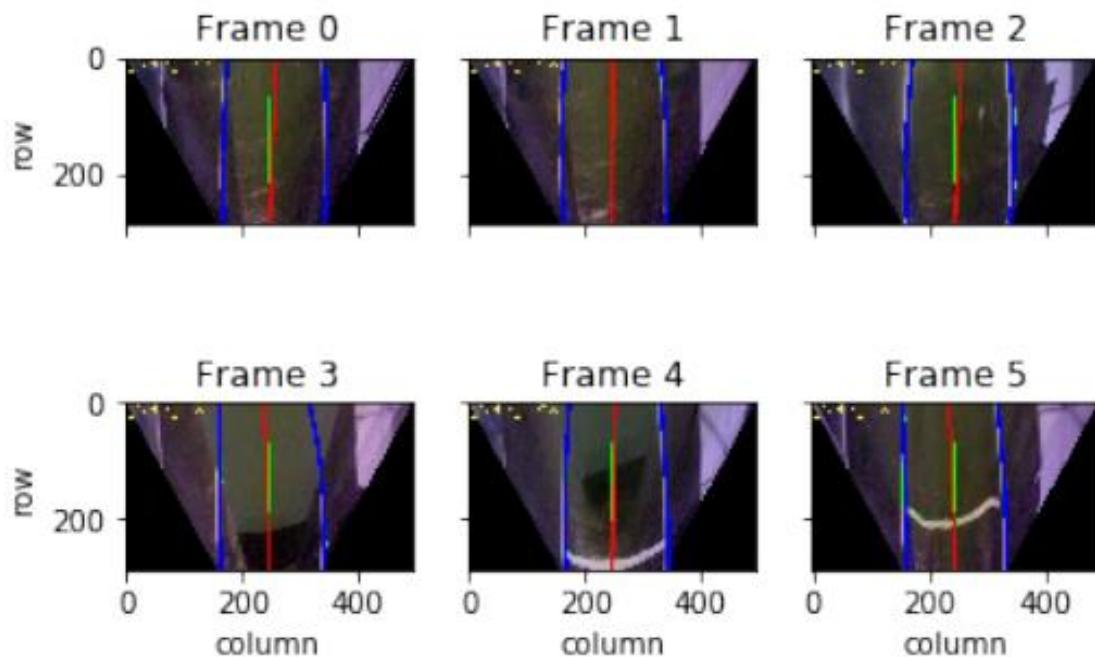


Figura 4.8. Líneas obtenidas a partir de los modelos de la sección recta.

4.2 Secciones curvas A y B

En las figuras 4.9 y la figura 4.10 tenemos seis diferentes imágenes capturas a lo largo de las secciones curvas A y B de la pista, las cuales ya han sido preprocesadas. A estas imágenes preprocesadas se les aplica el algoritmo de la búsqueda por ventana deslizante para agrupar los puntos pertenecientes a cada línea del carril en estas secciones curvas.

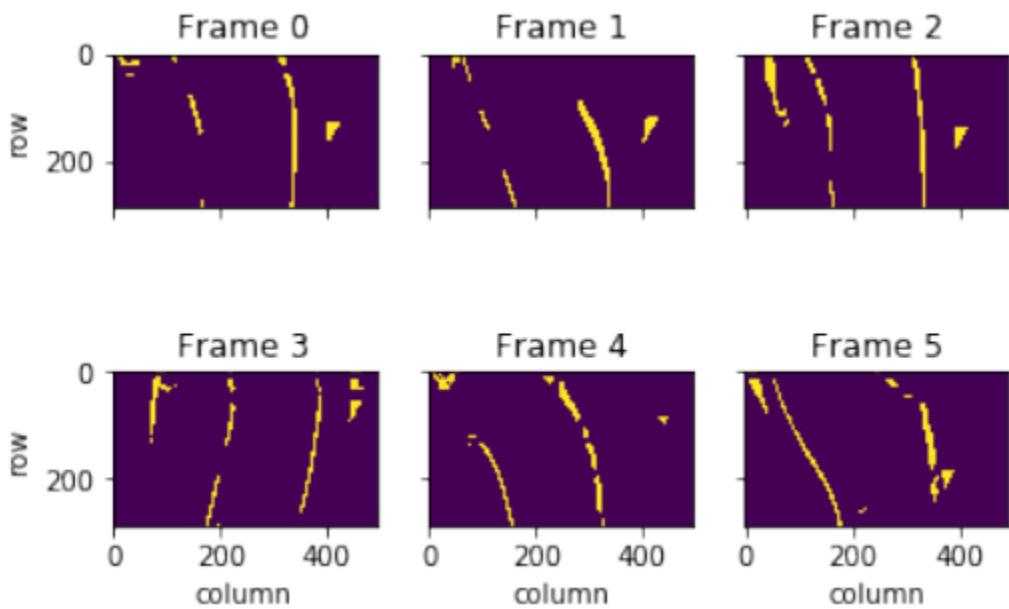


Figura 4.9. Imágenes preprocesadas de la sección curva A.

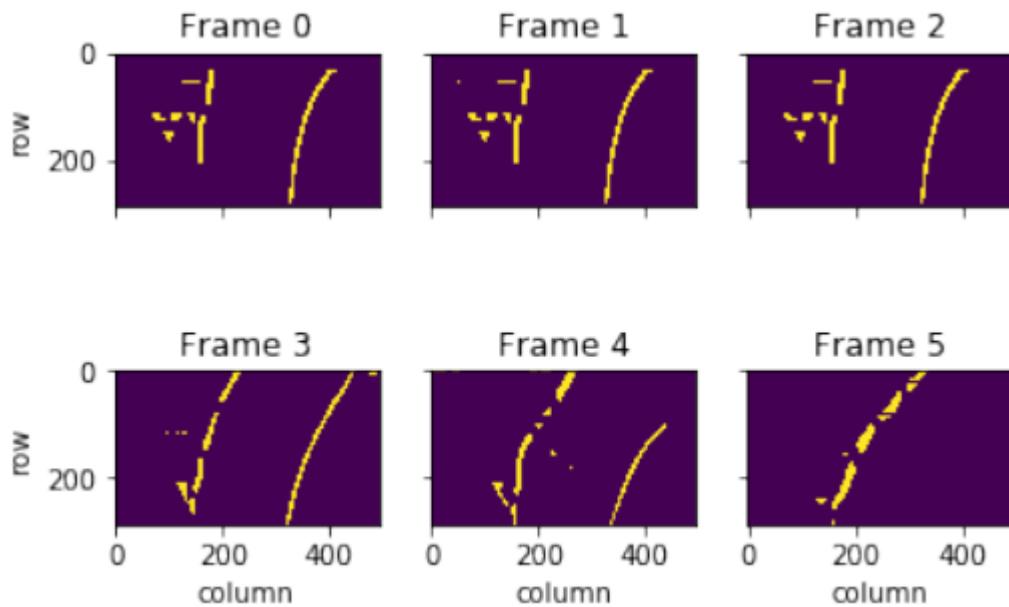


Figura 4.10. Imágenes preprocesadas de la sección curva B.

De las búsquedas realizadas tanto en la sección curva A y B, obtenemos un conjunto de puntos por muestra pertenecientes a las líneas izquierdas y derechas del carril en cada sección curva, en la figura 4.11 se visualizan las posiciones de los puntos agrupados en el frame para las líneas izquierdas de la curva A, en la figura

4.12 se visualizan las posiciones en el frame muestra de los puntos de las líneas derechas de la curva A, en la figura 4.13 se visualizan las posiciones de puntos encontrados para las líneas izquierdas de la curva B, en la figura 4.14 se visualizan las posiciones de los puntos de las líneas derechas de la curva B.

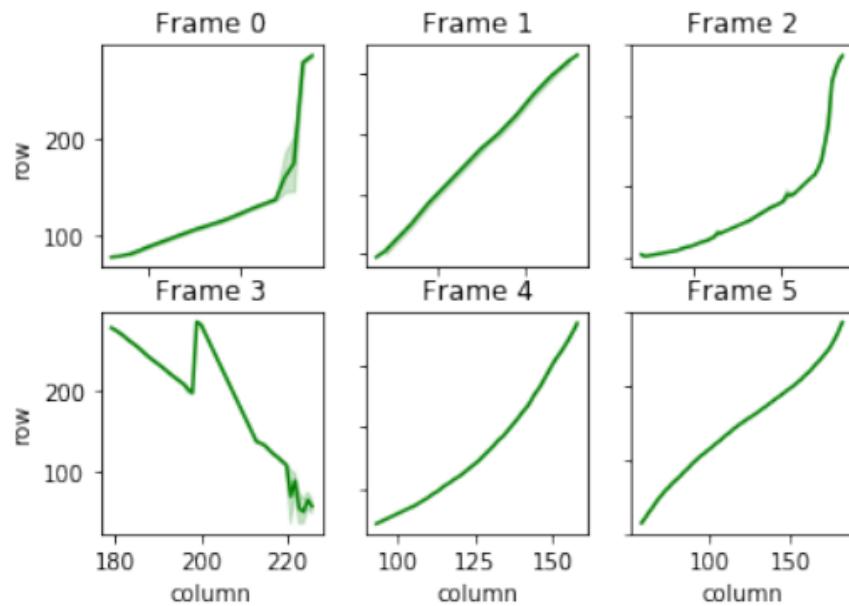


Figura 4.11. Posiciones en la imagen muestra de los puntos encontrados de las líneas izquierdas en la sección curva A.

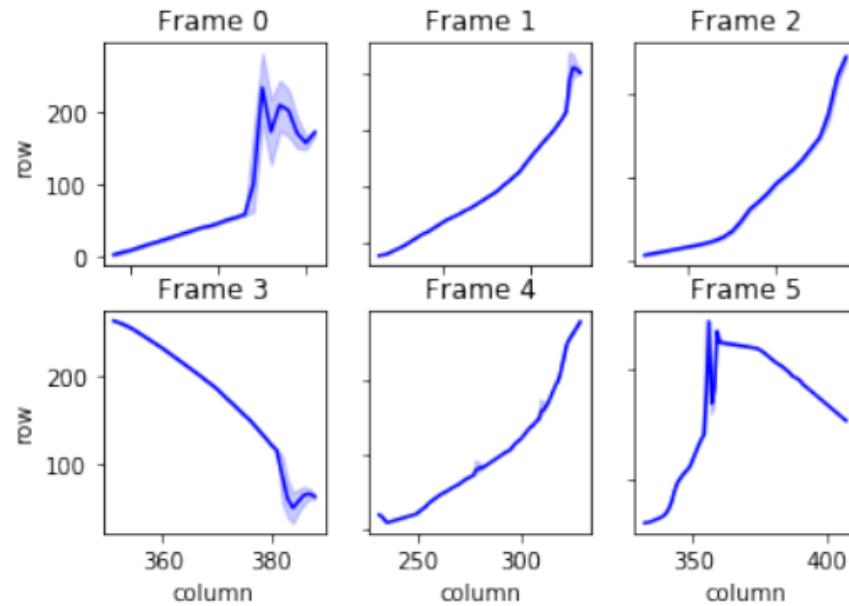


Figura 4.12. Posiciones en la imagen muestra de los puntos de las líneas derechas en la sección curva A.

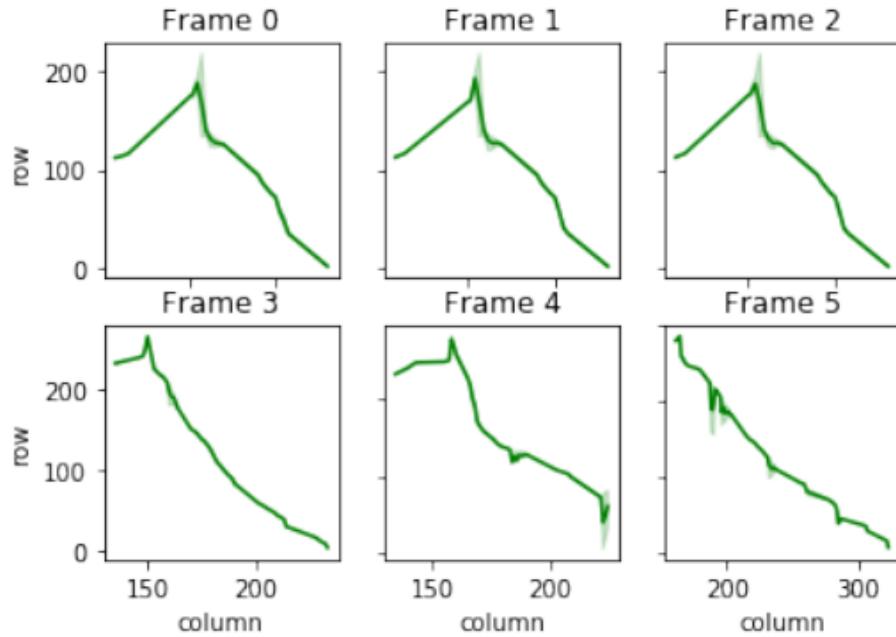


Figura 4.13. Posiciones en la imagen muestra de los puntos encontrados de las líneas izquierdas en la sección curva B.

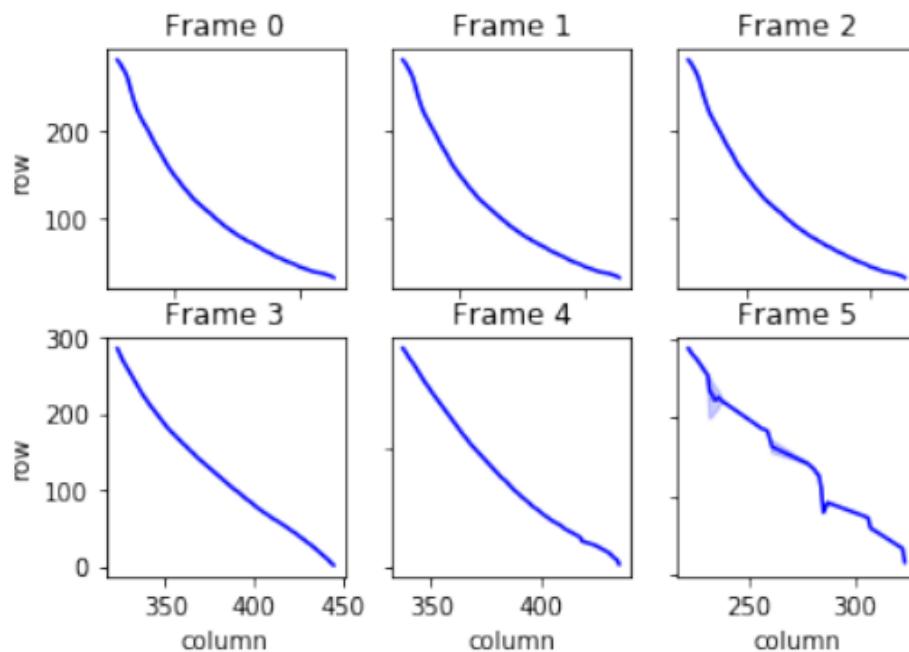


Figura 4.14. Posiciones en la imagen muestra de los puntos encontrados de las líneas derechas en la sección curva B.

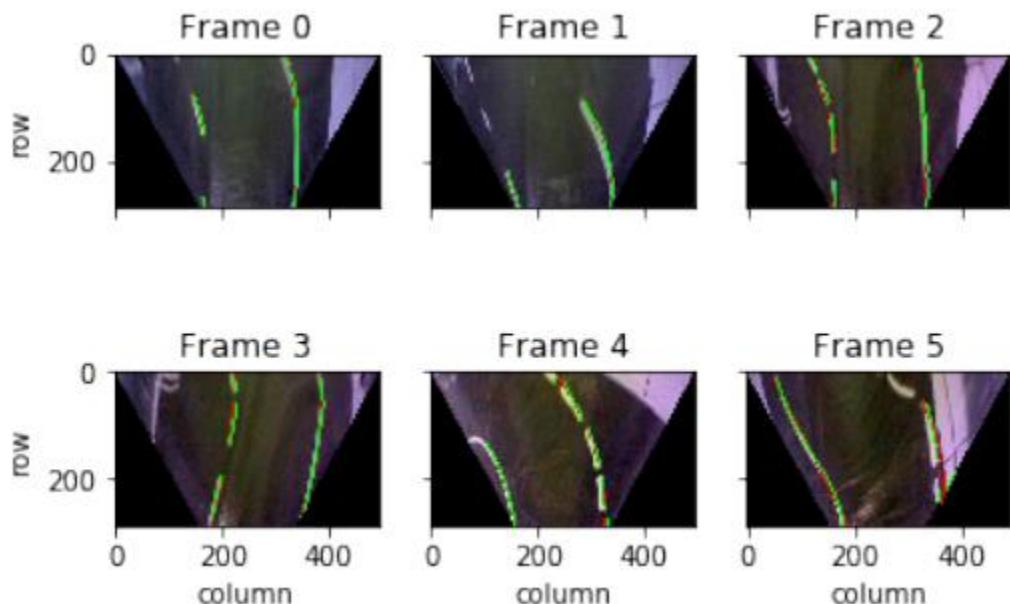


Figura 4.15. Conjuntos de puntos agrupados que pertenecen a las líneas del carril de la sección curva A.

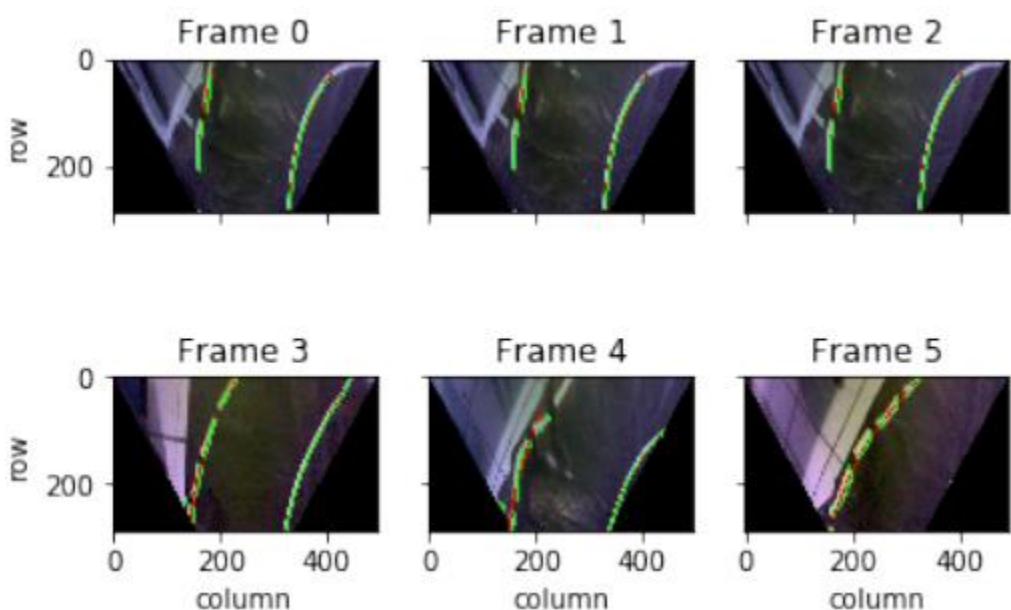


Figura 4.16. Conjuntos de puntos agrupados que pertenecen a las líneas del carril de la sección curva B.

En la figura 4.15 se observan los conjuntos de puntos agrupados sobre las líneas izquierdas y derechas del carril en la sección curva A. En la figura 4.16 están los conjuntos de puntos agrupados sobre las líneas izquierdas y derechas del carril en la sección curva B. De dichos conjuntos de datos podemos obtener las

siguientes estadísticas descriptivas, dispersiones estándares y medianas, ver tabla 10 para estadísticas de la sección curva A y tabla 11 para estadísticas de la sección curva B.

Tabla 10. Desviaciones estándares y medianas de las columnas de los puntos agrupados alrededor de las líneas del carril en la sección curva A.

Número de frame.	Mediana de la línea izquierda (posición en columna).	Desviación estándar de la línea izquierda. (posición en columna)	Mediana de la línea derecha (posición en columna).	Desviación estándar de la línea derecha. (posición en columna)
0	159.41052	6.49290	336.75555	5.58233
1	154.32352	6.529217	322.50251	16.55108
2	153.61463	14.81894	332.53900	5.64432
3	206.61928	16.52234	374.92712	10.67227
4	136.76470	15.43646	295.70995	28.16813
5	121.61538	40.53563	357.55900	15.70015

Tabla 11. Desviaciones estándares y medianas de las columnas de los puntos agrupados alrededor de las líneas del carril en la sección curva B.

Número de frame.	Mediana de la línea izquierda (posición en columna).	Desviación estándar de la línea izquierda. (posición en columna)	Mediana de la línea derecha (posición en columna).	Desviación estándar de la línea derecha. (posición en columna)
0	169.43529	9.95572	354.89959	22.65052
1	169.45882	9.87992	354.89156	22.66253
2	169.41176	9.91706	354.89558	22.64545
3	178.45451	26.86256	374.00349	37.28396
4	174.24623	21.99462	378.39247	28.94733
5	231.33195	47.90713	269.45238	33.67812

En las tablas 10 y 11 tenemos las columnas promedio y de la desviación estándares de la columna promedio de todos los puntos agrupados alrededor de las líneas izquierdas y derecha en la sección curva A y B. En las figuras 4.17, 4.18, 4.19 y 4.20 que muestran las distribuciones de puntos agrupados en función de su posición de columna en la imagen muestra para nuestras secciones curvas A y B, las distribuciones de los puntos encontrados en las secciones curvas presentan una mayor dispersión de sus puntos pertenecientes a las líneas del carril, en comparación de las dispersiones obtenidas en la sección recta.

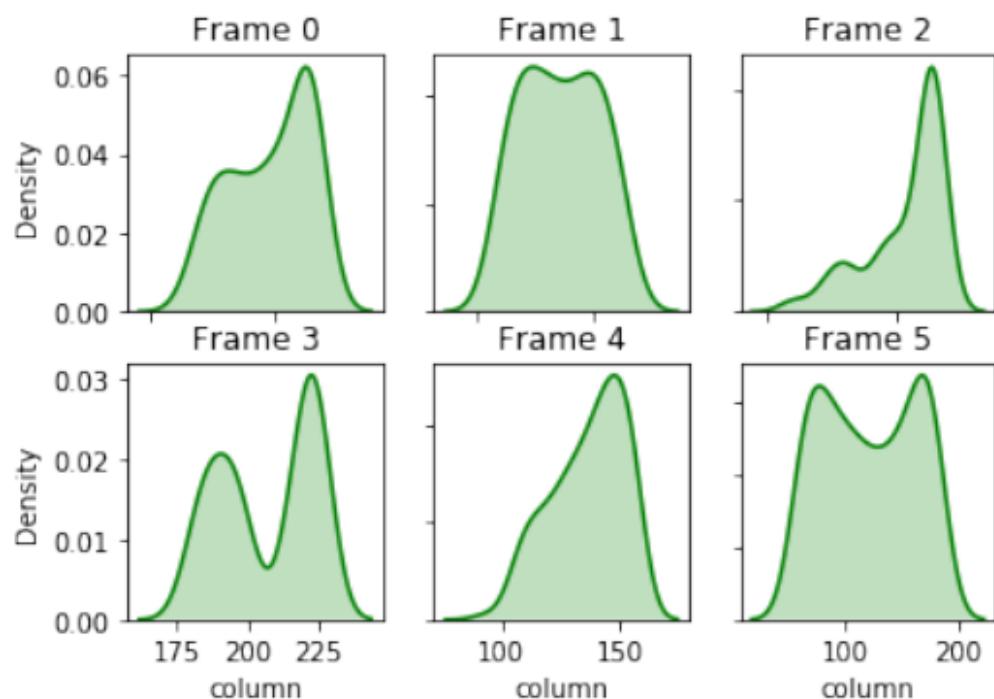


Figura 4.17. Distribución de puntos agrupados de las líneas izquierdas en la sección curva A.

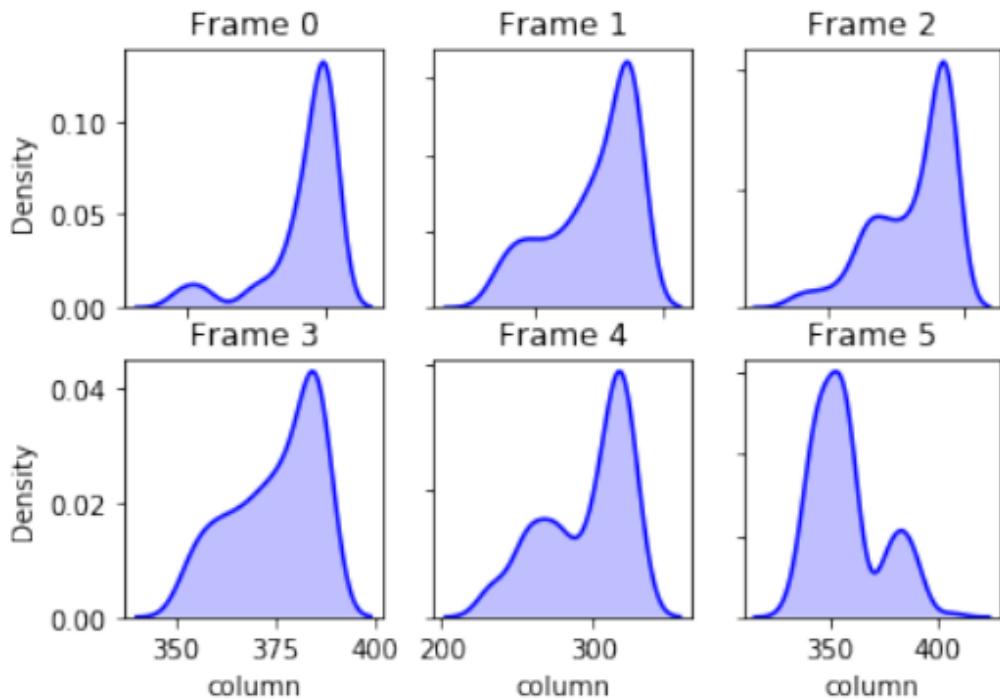


Figura 4.18. Distribución de puntos agrupados de las líneas derechas en la sección curva A.

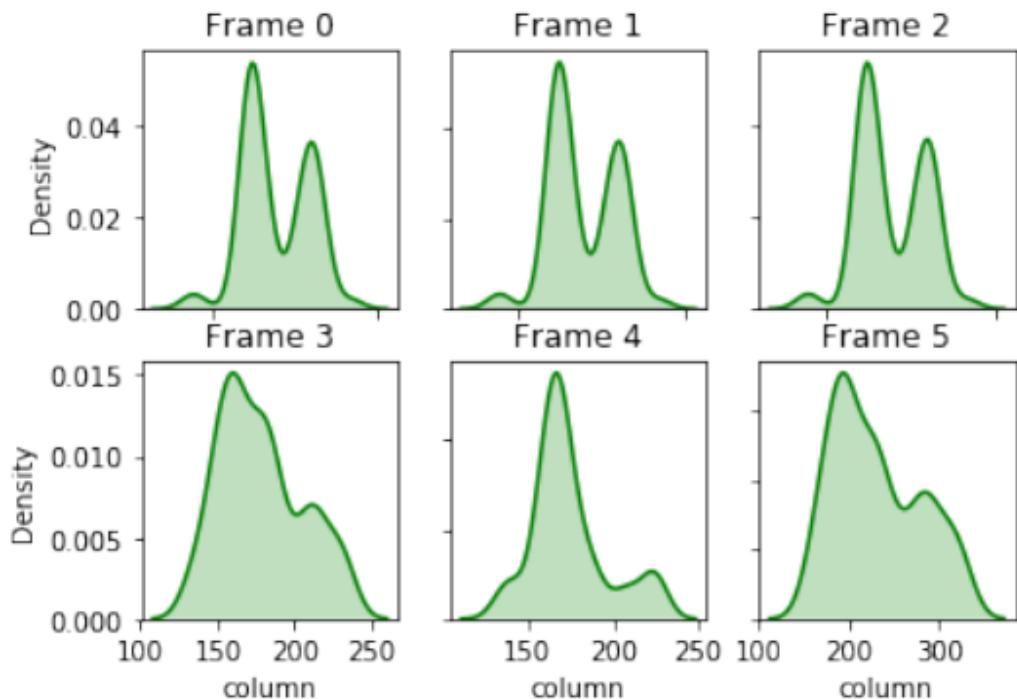


Figura 4.19. Distribución de puntos agrupados de las líneas izquierdas en la sección curva B.

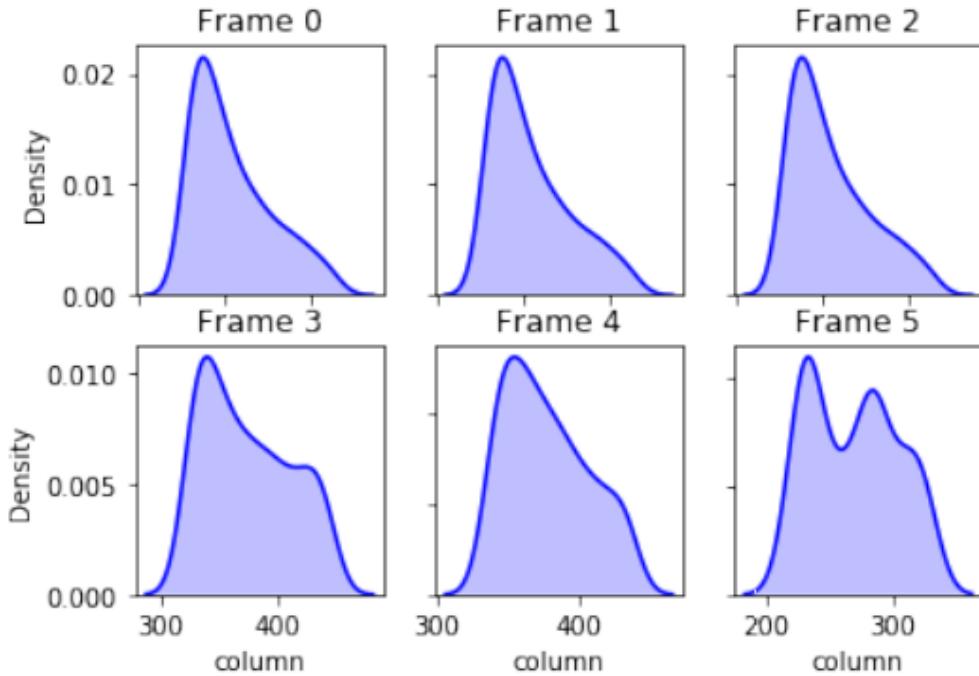


Figura 4.20. Distribución de puntos agrupados de las líneas derechas en la sección curva B.

A partir de los puntos agrupados se obtiene los modelos de las líneas izquierdas y derechas del carril, por medio regresiones que ajustan polinomios de segundo grado sobre el conjunto de puntos agrupados por cada línea del carril en las secciones curvas A y B. Para evaluar los modelos de regresión conseguidos de las secciones curvas, empleamos también las métricas del error absoluto medio (MAE) y la raíz del error cuadrático medio (RMSE). En las tablas 12 y 13 se muestran los errores RMSE y MAE de los modelos que describen las líneas izquierdas y derechas del carril para cada frame muestra tomado de las secciones curvas A y B.

Tabla 12. Errores de los modelos de las líneas del carril en la sección curva A.

Frame	MAE_LEFT (columnas)	RMSE_LEFT (columnas)	MAE_RIGHT (columnas)	RMSE_RIGHT (columnas)
0	0.694737	0.984084	1.310923	0.970370
1	0.485294	0.696631	0.823646	0.577889
2	2.058537	2.543284	1.012335	0.691489
3	3.182741	4.773552	1.358732	0.931174
4	0.790850	1.069482	2.891995	2.181818
5	2.981685	3.473605	9.527013	7.732919

Tabla 13. Errores de los modelos de las líneas del carril en la sección curva B.

Frame	MAE_LEFT (columnas)	RMSE_LEFT (columnas)	MAE_RIGHT (columnas)	RMSE_RIGHT (columnas)
0	2.558824	4.675971	1.763052	2.257518
1	2.576471	4.657694	1.779116	2.266395
2	2.529412	4.639980	1.775100	2.26550
3	3.120833	4.485625	0.825175	1.131247
4	7.030151	9.033995	1.188172	1.479247
5	4.680498	6.037916	4.388889	5.421152

En la figura 4.21 y 4.22 se muestra las líneas del carril creadas por medio de los polinomios de segundo grado obtenidos en las secciones curvas A y B.

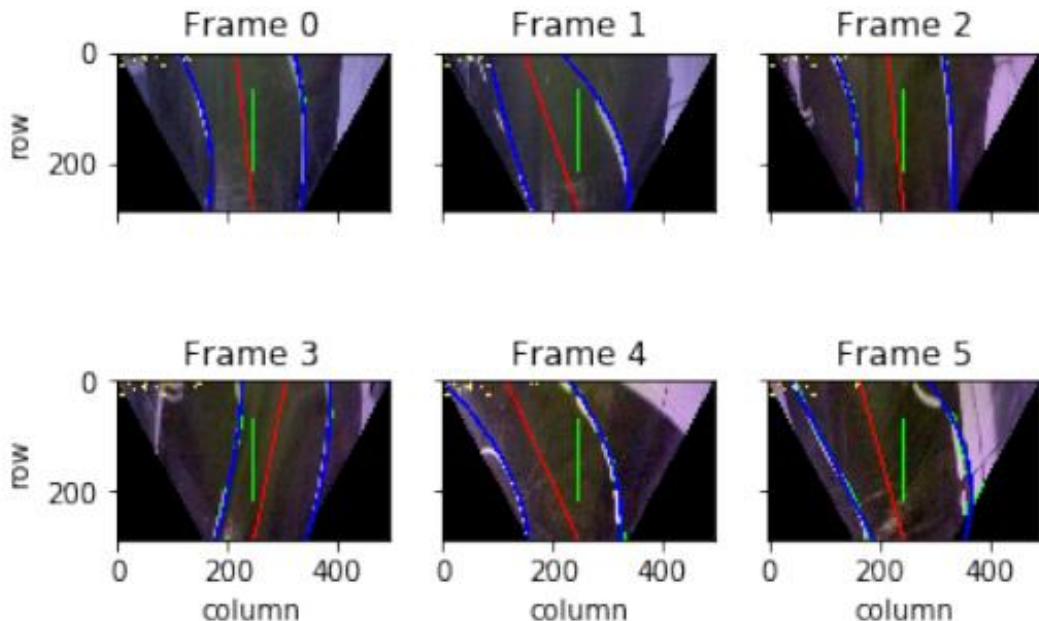


Figura 4.21. Líneas obtenidas a partir de los modelos de la sección curva A.

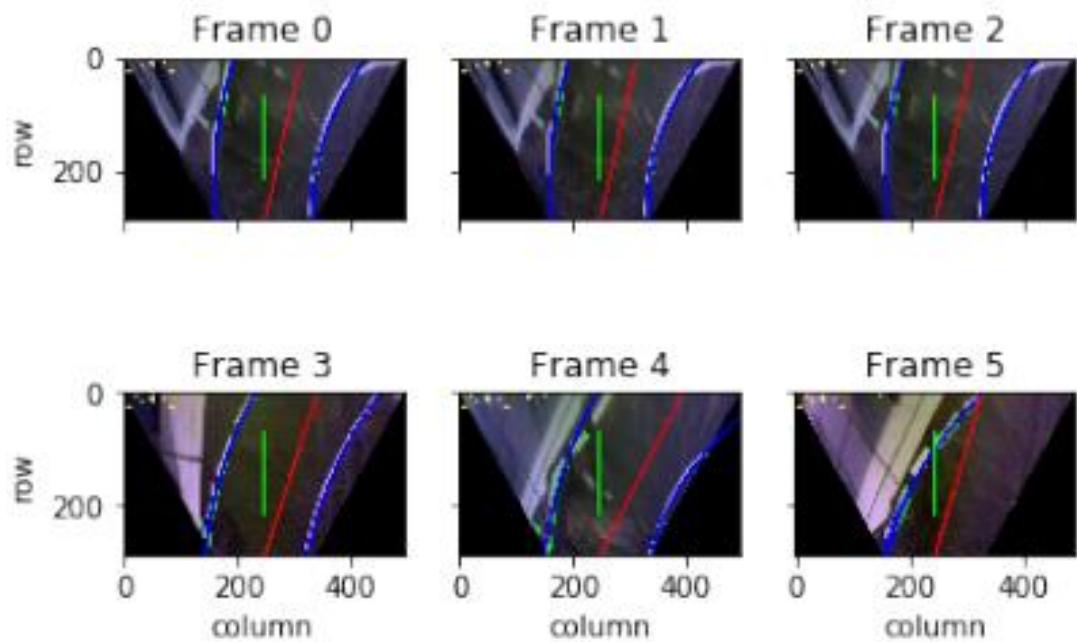


Figura 4.22. Líneas obtenidas a partir de los modelos de la sección curva B.

4.3 Sección intersección

En la figura 4.23 tenemos seis diferentes imágenes capturas a lo largo de la sección de intersección rotadas 90° y preprocesadas, a las cuales se les aplica el algoritmo de la búsqueda por ventana deslizante para agrupar los puntos pertenecientes a la línea de intersección.

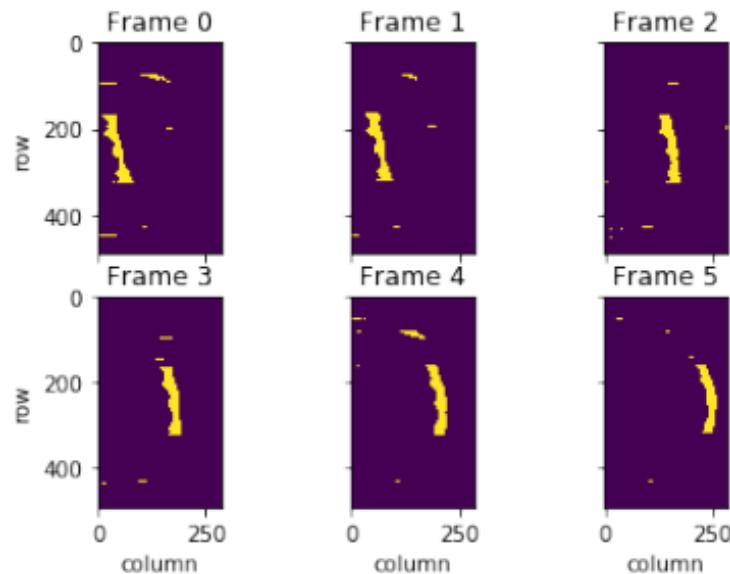
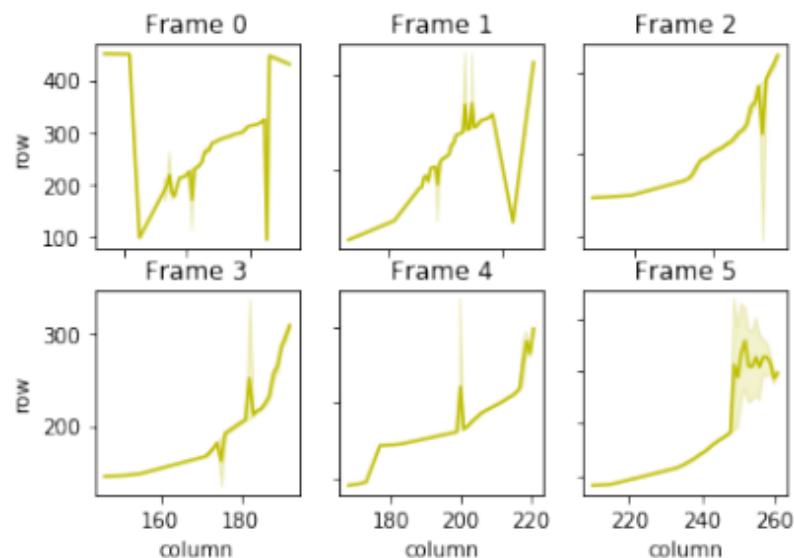


Figura 4.23. Imágenes preprocesadas de la sección de intersección.

De las

búsquedas verticales realizadas desde la fila 0 hasta la fila 492 de cada imagen muestra que se tomó de la sección intersección, obtenemos un conjunto de puntos por muestra pertenecientes a las líneas de la intersección, en la figura 4.24 se visualizan las posiciones en el frame muestra de puntos encontrados para las líneas

de las



intersecciones.

Figura 4.24. Posiciones en la imagen muestra de los puntos encontrados de las líneas en la sección de intersección.

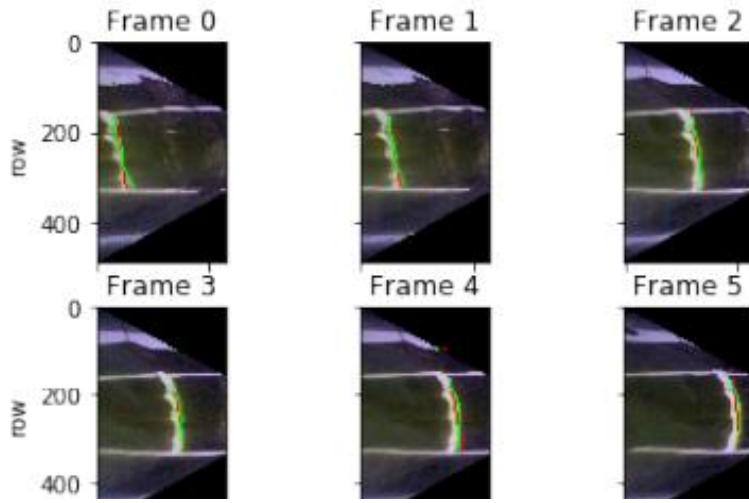


Figura 4.25. Conjuntos de puntos agrupados que pertenecen a las líneas de la sección de intersección.

En la figura 4.25 se observan los conjuntos de puntos agrupados sobre las líneas de la intersección. De dichos conjuntos de datos podemos obteneremos las siguientes estadísticas ver la tabla 14.

Tabla 14. Desviaciones estándares y medianas de las columnas de los puntos agrupados alrededor de las líneas del carril en la sección.

Número de frame.	Mediana de la línea de intersección (posición en columna).	Desviación estándar de la línea de la intersección. (posición en columna).
0	56.38095	12.64525
1	78.04166	11.07515
2	166.76687	8.72171
3	184.15151	8.26720
4	212.95348	11.58327
5	252.87878	8.755111

De la tabla 14 tenemos las columnas promedio de todos los puntos agrupados alrededor de las líneas de intersección de cada imagen muestra, también se muestran las desviaciones estándares que presentan las agrupaciones de las

columnas de los puntos por frame en la sección de intersección. Para realizar una observación visual de la distribución de los puntos agrupados alrededor de las líneas de la sección de intersección de cada imagen muestra, se tiene la figura 4.26 que presenta la distribución de puntos agrupados en función de su posición de columna en el frame muestra, al ser una línea de puntos, la distribución estos puntos se encuentra en un rango pequeño de aproximadamente 30 columnas sin tomar en cuenta los valores extremos de la distribución.

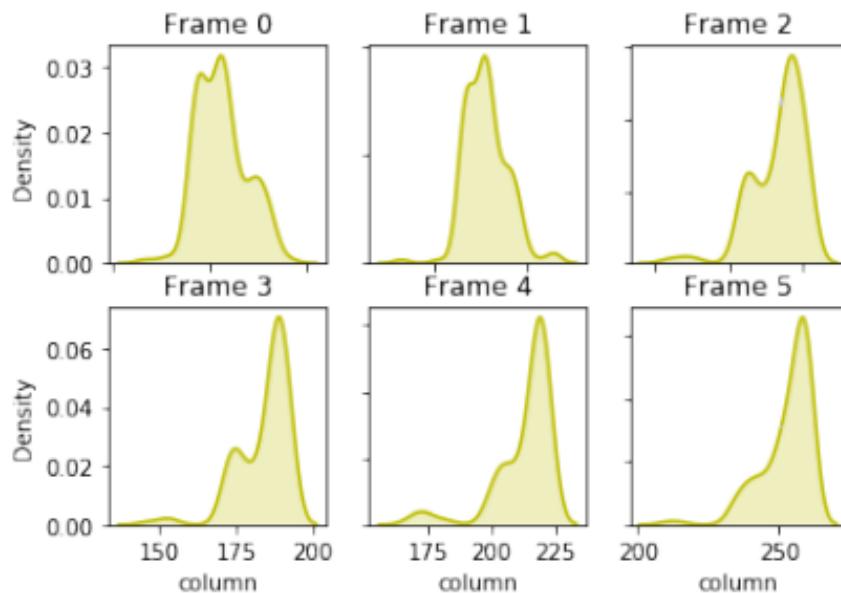


Figura 4.26. Distribución de puntos agrupados de las líneas de la sección de intersección.

Por medio de los puntos agrupados se obtienen los modelos de las líneas de la intersección, por medio de regresiones lineales que ajustan rectas sobre el conjunto de puntos agrupados por cada línea de intersección. Para evaluar los modelos se emplean las métricas del error absoluto medio (MAE) y la raíz del error cuadrático medio (RMSE). En la tabla 15 se muestran los errores RMSE y MAE de los modelos que describen las líneas en la sección de intersección.

Tabla 15. Errores de los modelos de las líneas de la sección de intersección.

Frame	MAE_CROSSING (columnas)	RMSE_CROSSING (columnas)
0	5.857143	10.186359
1	2.946429	5.855909

2	2.165644	4.127567
3	2.666667	4.314879
4	4.860465	6.583983
5	5.515152	6.776966

En la figura 4.27 se muestran las líneas generadas a partir de los modelos obtenidos en la sección de la intersección.

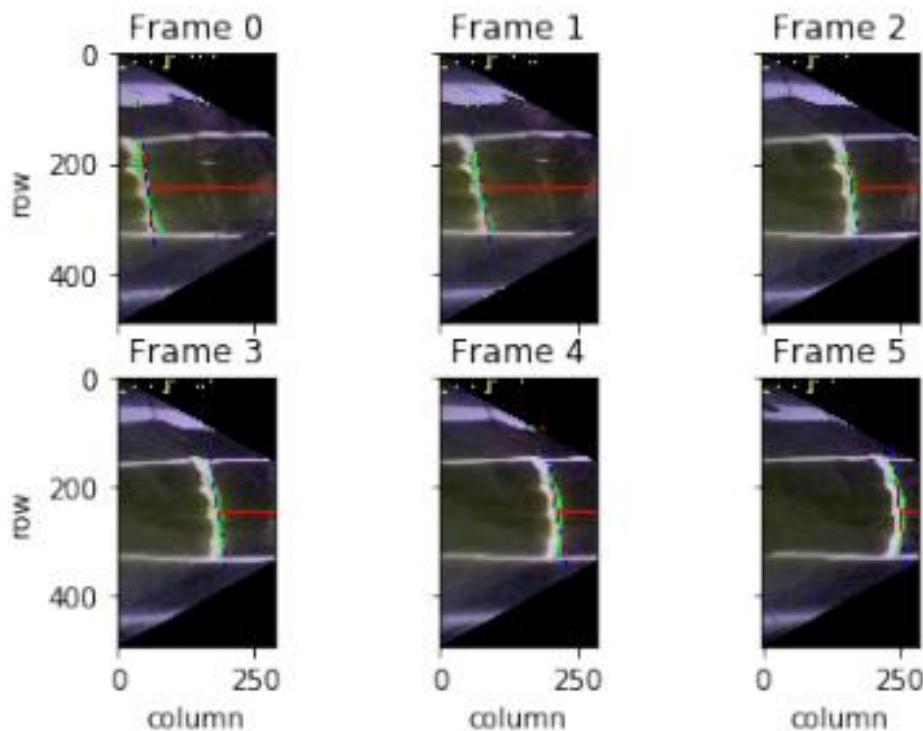


Figura 4.27. Líneas obtenidas a partir de los modelos de la sección de intersección.

4.4 Comparativa del desempeño de los modelos de las líneas del carril e intersecciones

Para poder realizar una comparación del desempeño general de los modelos de las líneas izquierdas y derechas del carril por cada sección de pruebas, calculamos un valor promedio del error RMSE y MAE de todos los modelos encontrados por línea del carril en una determinada sección de pruebas. En la sección recta tenemos un MAE promedio de 1.211 columnas y un RMSE promedio de 2.025 columnas para las líneas izquierdas del carril, un MAE promedio de 0.598 columnas y un RMSE

promedio de 0.847 columnas para las líneas derechas del carril. En la sección curva calculamos un MAE promedio de 1.699 columnas y un RMSE promedio de 2.256 columnas para las líneas izquierdas del carril, un MAE promedio de 2.181 columnas y un RMSE promedio de 2.821 columnas para las líneas derechas del carril. En la sección curva B obtuvimos un MAE promedio de 3.748 columnas y un RMSE promedio de 5.587 columnas las para líneas izquierdas del carril, un MAE promedio de 1.953 columnas y un RMSE promedio de 2.469 columnas las para líneas derechas del carril. En la figura 4.28 se muestra un gráfico comparativa de los errores de los modelos generados para las líneas del carril por secciones de la pista.

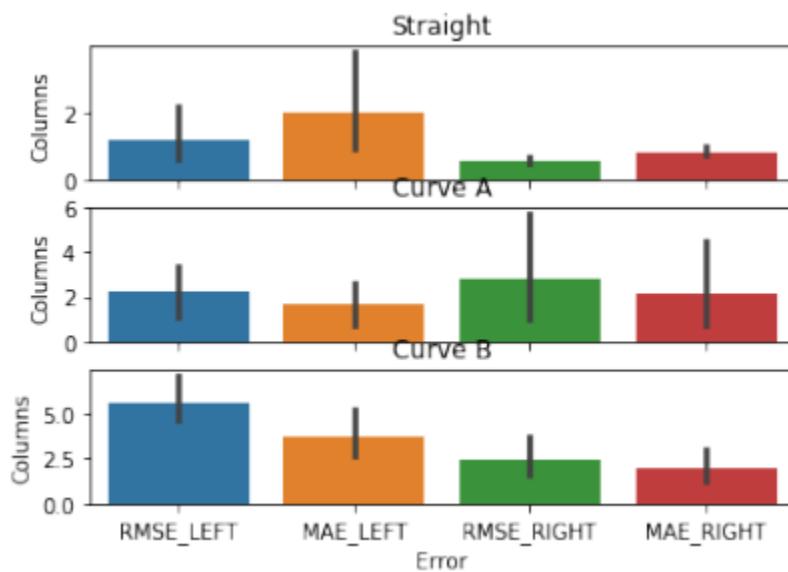


Figura 4.28. Comparación de errores de los modelos de las líneas del carril.

De la figura 4.28 se observa que los modelos de las líneas izquierdas presentan en promedio un mayor error en sus pronósticos que los modelos de las líneas derechas, los modelos con los menores errores promedio son los de la sección recta, en su contraparte los modelos con los mayores errores promedio son los de la sección curva B.

Para la sección de intersección obtuvimos un MAE promedio de 4.0012 columnas y un RMSE promedio de 6.306 columnas las para líneas de las intersecciones. Esta es la sección de pruebas que presenta los errores promedio más grandes.

4.5 Frecuencias de procesamiento de los nodos del subsistema de visión artificial

En total el subsistema de visión artificial cuenta con tres nodos, cada nodo corre en un único hilo de ejecución. El primer nodo "/jetson_camera/raw", es el encargado de publicar las imágenes capturadas por la cámara realizando solamente un redimensionamiento de la imagen capturada. El segundo nodo "lane_detection", es el encargado de la detección de las líneas del carril, este nodo publica la posición y dirección del coche a escala calculada a partir los modelos que describen las líneas del carril. El tercer nodo "crossing_detection", se encarga de la detección de las líneas de intersección, este nodo publica la distancia desde la intersección hasta el coche a escala. Para analizar la velocidad de ejecución de cada nodo se evalúa la frecuencia de publicación de cada nodo, la frecuencia de publicación de cada nodo varía en un rango de frecuencias mostrado en la tabla 16.

Tabla 16. Frecuencias de publicación de cada nodo del subsistema de visión artificial.

Nodo	Frecuencia mínima de publicación (Hz).	Frecuencia máxima de publicación (Hz).
"/jetson_camera/raw"	25	40
"lane_detection"	16	34
"crossing_detection"	19	37

Capítulo 5 Conclusiones y trabajos futuros

Este capítulo concentra las conclusiones más relevantes obtenidas a partir del diseño de la arquitectura del sistema tipo AutoModelCar, así como el diseño e implementación del sistema de visión artificial para percepción de la pista de carreras de la categoría AutoModelCar. También se proponen algunas mejoras a realizar para la continuación de este proyecto.

5.1 Conclusiones

Las principales conclusiones generadas a partir del diseño de la arquitectura y del diseño e implementación del subsistema de visión artificial son:

- El diseño de la arquitectura robótica que se ha planteado es jerárquico y modular, en esta arquitectura cada componente del sistema está pensado como un módulo que se puede integrar de manera independiente de la implementación de otros módulos con los que interacciona, por lo que esta arquitectura es capaz de soportar la integración de varios componentes de software y hardware propuestos para realizar las misiones de la categoría AutoModelCar del Torneo Mexicano de Robótica. Al ser modular, el sistema es flexible permitiendo la implementación de nuevos módulos o mejorar los módulos propuestos para implementar en esta arquitectura.
- El subsistema de visión artificial diseñado e implementado está compuesto de tres módulos o nodos, al emplear una integración de los tres nodos con ROS, se agilizo y facilito el proceso de implementación, así como la depuración de los nodos, esto debido principalmente al factor de trabajar cada nodo de forma independiente y no en un solo programa.
- El primer nodo "/jetson_camera/raw" es capaz de capturar una imagen desde la cámara redimensionarla y publicarla con una frecuencia de 25 a 40 Hz. El segundo nodo "lane_detection" probó ser robusto en diversas secciones de la pista de pruebas para la detección de las líneas de carril haciendo uso solamente de las técnicas clásicas de visión artificial basadas en modelos, esto se puede comprobar con los modelos obtenidos para describir las líneas

del carril, los cuales tiene pequeños errores en la estimación de la columna de la línea del carril dada una fila específica ; además se ser nodo con una rápida ejecución, logrando publicar los datos de la posición y dirección del coche a escala respecto a las líneas del carril en un rango de frecuencias de 16 a 34 Hz, por lo que el nodo logra realizar el procesamiento completo de un frame entre 0.029 s y 0.0625 s . El segundo nodo "crossing_detection" encargado de la detección de las intersecciones también probó ser robusto, a pesar de ser la sección con los mayores errores promedio en el pronóstico de sus modelos, los errores obtenidos aún son pequeños y bastante aceptables; la velocidad de ejecución este nodo es también rápida, logrando publicar la distancia entre el choche a escala y la línea del cruce en un rango de frecuencias de 19 a 37 Hz, por lo que este nodo realizar el procesamiento completo de un frame entre 0.027 s y 0.0526 s.

5.2 Trabajos futuros

Dados los resultados obtenidos, se realizan las siguientes propuestas para continuar y mejorar con el trabajo realizado:

- Se propone incorporar un algoritmo de clustering como el DBSCAN, para poder realizar una agrupación de puntos pertenecientes a las líneas del carril e intersección con una menor cantidad de ruido que los puntos agrupados por el algoritmo de la búsqueda por ventana deslizante, de esta forma reducir el error en los modelos de las líneas del carril e intersecciones.
- Incorporar un módulo de detección de obstáculos por visión computacional a la arquitectura robótica, para lograr una redundancia en la detección de obstáculos en la pista y no solo se realice con los sensores ultrasónicos propuestos.
- Sustituir la cámara actual por una cámara con una mayor apertura focal, para tener una mejor imagen de la pista de la competencia.

Referencias

- [1] F. García & E. Arrieta, "Coche autónomo: las empresas que harán posible el fin de los accidentes", *expansion.com*, 2017. [En línea]. Disponible en: <https://www.expansion.com/economia-digital/innovacion/2017/08/12/5989f28746163f090f8b462b.html>. [Accedido: 12-enero-2020].
- [2] N. Martínez Losa, "Las distracciones son la segunda causa de accidentes mortales en España", *Cea-online.es*, 2008. [En línea]. Disponible en: https://www.cea-online.es/prensa/e-cea/Mag_204_distracciones.html. [Accedido: 14-enero-2020].
- [3] "Dona Alemania vehículos autónomos a equipos de investigación mexicanos", *Alianzaautomotriz.com*, 2016. [En línea]. Disponible en: <https://alianzaautomotriz.com/dona-alemania-vehiculos-autonomos-a-equipos-de-investigacion-mexicanos/>. [Accedido: 14-enero-2020].
- [4] "AutoModelCar - TMR2020", *femexrobotica.org*, 2019. [En línea] Disponible en: <https://www.femexrobotica.org/tmr2020/automodelcar/>. [Accedido 16-enero-2020].
- [5] "Autopilot", *tesla.com*, 2020. [En línea] Disponible en: https://www.tesla.com/es_MX/autopilot. [Accedido: 16-enero-2020].
- [6] "Technology", *waymo.com*. [En línea] Disponible en: <https://waymo.com/tech/>. [Accedido: 17-enero-2020].
- [7] J. Rojo & R. Rojas, "Spirit of Berlin: An Autonomous Car for the DARPA Urban Challenge Hardware and Software Architecture", *Citeseerx.ist.psu.edu*, 2007. [En línea] Disponible en: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.3075&rep=rep1&type=pdf>. [Accedido: 18-enero-2020].

- [8] D. Pomerleau, "RALPH: Rapidly Adapting Lateral Position Handler", *Ri.cmu.edu*. [En línea] Disponible en https://www.ri.cmu.edu/pub_files/pub2/pomerleau_dean_1995_2/pomerleau_dean_1995_2.pdf. [Accedido: 18- enero- 2020].
- [9] T. Jochem, "Back to the Future: 1995 Robomobile", *sudonull.com*, 2015. [En línea] Disponible en: <https://sudonull.com/post/56199-Back-to-the-Future-1995-Robomobile>. [Accedido: 19-enero- 2020].
- [10] S. García, "Construye ingeniero mexicano autos robot en Berlín", *sipse.com*, 2014. [En línea] Disponible en: <https://sipse.com/mexico/cientifico-mexicano-raul-rojas-disena-autos-inteligentes-en-alemania-92463.html>. [Accedido: 19-enero- 2020].
- [11] H. Buchheister & G. Dick, "Projekt 27: Carolo-Cup", *wiki.hshl.de*, 2016. [En línea] Disponible en: https://wiki.hshl.de/wiki/index.php/Projekt_27:_Carolo-Cup. [Accedido: 25-enero- 2020].
- [12] I. Karouach & S. Ivanov, "Lane Detection and Following Approach in Self-Driving Miniature Vehicles", University of Gothenburg, 2016.
- [13] CaroloCup TuBs, *Carolo-Cup 2017 - Rundkurs mit Hindernissen: Team Spatzenhirn*. 2017.
- [14] R. Zuccolo, "Self-driving Cars — Advanced computer vision with OpenCV, finding lane lines", *chatbotslife.com*, 2017. [En línea] Disponible en: <https://chatbotslife.com/self-driving-cars-advanced-computer-vision-with-opencv-finding-lane-lines-488a411b2c3d>. [Accedido: 02-febrero- 2020].
- [15] "Unsere Softwarestruktur", *Kitcar-team.de*, 2020. [En línea] Disponible en: <https://kitcar-team.de/software.php>. [Accedido: 05-febrero- 2020].
- [16]"OpenCV", *Opencv.org*, 2020. [En línea] Disponible en: <https://opencv.org/about/>. [Accedido: 06-febrero- 2020].
- [17]"Miscellaneous Image Transformations", *Docs.opencv.org*. [En línea] Disponible en: https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html. [Accedido: 09-febrero- 2020].

- [18] R. Murphy, *Introduction to AI robotics*. Cambridge, pp. 1-40, MA: MIT Press, 2005.
- [19] Y. Pyo, H. Cho & R. Jung, *ROS Robot Programming*, 1ra ed., pp. 1-59, Londres: ROBOTIS Co.,Ltd., 2017.
- [20] A. Mahtani, L. Sánchez, E. Fernández & A. Martínez, *Effective robotics programming with ROS*, 3ra ed., pp. 1-100, Birmingham, UK: Packt Publishing Ltd, 2016.
- [21] W. Newman, *A systematic approach to learning robot programming with ROS.*, pp. 1-53, Boca Raton: CRC Press, Taylor & Francis Group, 2018.
- [22] S. Sánchez Tirado, *Integración de Técnicas de Manejo de la Imagen Digital para la Web y Aplicaciones Multimedia*, pp. 3-74, 2006.
- [23] "OpenCV: Color conversions", *Docs.opencv.org*. [En línea]. Disponible en: https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html. [Accedido: 1-marzo- 2020].
- [24] "Tutorial 1: Image Filtering", *Ai.stanford.edu*. [En línea]. Disponible en: <https://ai.stanford.edu/~syueung/cvweb/tutorial1.html>. [Accedido: 2- marzo- 2020].
- [25] R. Szeliski, *Computer vision*, pp. 111-122, New York: Springer, 2010.
- [26] A. Mikdad Muad, A. Hussain & S. Abdul Samad, "Implementation of Inverse Perspective Mapping Algorithm for the Development of an Automatic Lane Tracking System", Universiti Kebangsaan Malaysia.
- [27] "Histograms", *Ncss.com*. [En línea]. Disponible en: <https://www.ncss.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Histograms.pdf>. [Accedido: 15- marzo- 2020].
- [28] "Learning algorithm", *Riptutorial.com*. [En línea]. Disponible en: <https://riptutorial.com/ebook/algorithm>. [Accedido: 20- marzo- 2020].
- [29] S. Chapra, R. Canale & J. Del Valle Sotelo, *Métodos numéricos para ingenieros*, 5th ed., pp. 249-272, 466-502, México: McGraw-Hill, 2007.

[30] "About", OpenCV.org. [En línea]. Disponible en: <https://opencv.org/about/>. [Accedido 30- marzo- 2020].

Apéndice A: jetson_camera.cpp.

```
#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <opencv2/opencv.hpp>
#include <cv_bridge/cv_bridge.h>
std::chrono::time_point<std::chrono::system_clock> start,end;
std::chrono::duration<double> elapsed_seconds;

std::string gstreamer_pipeline (int capture_width, int capture_height, int display_width, int display_height, int framerate, int flip_method)
{
    return "nvarguscamerasrc ! video/x-raw(memory:NVMM), width=(int)" + std::to_string(capture_width) + ", height=(int)" +
           std::to_string(capture_height) + ", format=(string)NV12, framerate=(fraction)" + std::to_string(framerate) +
           "/1 ! nvvidconv flip-method=" + std::to_string(flip_method) + " ! video/x-raw, width=(int)" + std::to_string(display_width) + ", height=(int)" +
           std::to_string(display_height) + ", format=(string)BGRx ! videoconvert ! video/x-raw, format=(string)BGR ! appsink";
}

void set_start_time()
{
    start = std::chrono::system_clock::now();
}
void set_end_time()
{
    end = std::chrono::system_clock::now();
}
int FPS_subscriber()
{
    elapsed_seconds = end-start;
    float t = elapsed_seconds.count();
    return 1/t;
}

int main(int argc, char** argv)
{
    ros::init(argc, argv, "jetson_camera");
    ros::NodeHandle nh;
```

```

image_transport::ImageTransport it(nh);
image_transport::Publisher pub = it.advertise("/jetson_camera/raw", 1);
sensor_msgs::ImagePtr msg;
int capture_width = 2464 ;
int capture_height = 1440 ;
int display_width = 492;
int display_height = 288;
int framerate = 60 ;
int flip_method = 0 ;

std::string pipeline = gstreamer_pipeline(capture_width,
capture_height,
display_width,
display_height,
framerate,
flip_method);
std::cout << "Using pipeline: \n\t" << pipeline << "\n";
cv::VideoCapture cap(pipeline, cv::CAP_GSTREAMER);
if(!cap.isOpened())
{
    ROS_ERROR("Failed to open camera\n");
    return (-1);
}
cv::Mat img;
while(true)
{
    set_start_time();
    if (!cap.read(img))
    {
        ROS_ERROR("Capture read error \n");
        break;
    }
    msg = cv_bridge::CvImage(std_msgs::Header(), "bgr8", img).toImageMsg();
    pub.publish(msg);
    set_end_time();
    ROS_INFO("[FPS]: %i ",FPS_subscriber());
}
cap.release();
return 0;
}

```

Apéndice B: lane_detection.cpp.

```
#include<iostream>
#include<math.h>
#include<string>
#include<ros/ros.h>
#include <std_msgs/String.h>
#include <std_msgs/Int16.h>
#include <std_msgs/UInt8.h>

/*Using image transport for publish and subscribing to image in ros,
allows subscriber to compressed image stream*/
#include <image_transport/image_transport.h>
/*Using the headers for CvBridge, to image encodings*/
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
/*Include the headers for opencv image processing and GUI modules */
#include <opencv2/opencv.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
//Library to use time of computer
#include <chrono>
#include <ctime>
using namespace cv;
using namespace std;

//static const std::string OPENCV_WINDOW = "Original";
static const std::string OPENCV_WINDOW = "LINES DETECTION";
class LineDetection
{
private:
    ros::NodeHandle nh_;
    image_transport::ImageTransport it_;
    image_transport::Subscriber image_sub_;
    ros::Publisher center_pub;
    ros::Publisher angle_line_pub;
    std_msgs::Int16 center_message;
    std_msgs::UInt8 angle_line_message;
    std::chrono::time_point<std::chrono::system_clock> start,end;
    std::chrono::duration<double> elapsed_seconds;
    stringstream ss;
    /*variable image*/
    Mat frame;
    Mat img_edges;
    Mat img_lines;
    Mat img_perspective;
```

```

vector<Point> left_points,right_points;
vector<Point> left_line,right_line,center_line;
float polyleft[3],polyright[3];
Point2f Source[4];
Point2f Destination[4];
int center_cam;
int center_lines;
int distance_center;
int angle;
public:
//Constructor
LineDetection():it_(nh_)
{
// Subscribe to input video feed and publish output video feed
image_sub_ = it_.subscribe("/jetson_camera/raw",1,&LineDetection::LineDetectionCb, this);
center_pub = nh_.advertise<std_msgs::Int16>("/distance_center_line",1000);
angle_line_pub = nh_.advertise<std_msgs::UInt8>("/angle_line_now",1000);

Source[0]=Point2f(784*0.2,676*0.2);
Source[1]=Point2f(1680*0.2,676*0.2);
Source[2]=Point2f(0*0.2,1200*0.2);
Source[3]=Point2f(2464*0.2,1200*0.2);
//init points to destination
Destination[0]=Point2f(784*0.2,0);
Destination[1]=Point2f(1680*0.2,0);
Destination[2]=Point2f(784*0.2,1440*0.2);
Destination[3]=Point2f(1680*0.2,1440*0.2);
distance_center = 0.0;
center_cam = 0;
center_lines = 0;
angle = 0;
}
//Callback function
void LineDetectionCb(const sensor_msgs::ImageConstPtr& msg)
{
set_start_time();
cv_bridge::CvImagePtr cv_ptr;
try
{
cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::BGR8);
copyImageRGB(cv_ptr->image,frame);
}
catch (cv_bridge::Exception& e)
{
}
}

```

```

        ROS_ERROR("cv_bridge exception: %s", e.what());
        return;
    }
    Perspective(frame);
    img_edges = Threshold(frame);
    width_filter(img_edges);
    locate_lanes(img_edges,frame);
    draw_lines(frame);
    center_message.data = distance_center;
    center_pub.publish(center_message);
    angle_line_message.data = angle;
    angle_line_pub.publish(angle_line_message);
    set_end_time();
    ROS_INFO("[FPS]: %i ",FPS_subscriber());
}
void set_start_time()
{
    start = std::chrono::system_clock::now();
}
void set_end_time()
{
    end = std::chrono::system_clock::now();
}
int FPS_subscriber()
{
    elapsed_seconds = end-start;
    float t = elapsed_seconds.count();
    return 1/t;
}
//Method to change image from BGR to RGB
void copyImageRGB(Mat & image_BGR,Mat &image_RGB)
{
    cvtColor(image_BGR, image_RGB, COLOR_BGR2RGB);
}
//Method to rotate image
void rotate(Mat &src, double angle=180.0
{
    Point2f pt(src.cols/2., src.rows/2.);
    Mat r = getRotationMatrix2D(pt, angle, 1.0);
    warpAffine(src, src, r, Size(src.cols, src.rows));
}
void Perspective(Mat &frame)
{
    Mat matrixPerspective( 2, 4, CV_32FC1 );
    matrixPerspective = Mat::zeros( frame.rows, frame.cols, frame.type() );

```

```

matrixPerspective = getPerspectiveTransform(Source, Destination);
warpPerspective(frame, frame, matrixPerspective, Size(frame.cols, frame.rows));
}

Mat Threshold(Mat frame)
{
    Mat frameThreshold;
    frame.convertTo(frame, -1, 1, -15);
    cvtColor(frame, frameThreshold, COLOR_RGB2GRAY);
    medianBlur(frameThreshold, frameThreshold, 5);
    erode(frameThreshold, frameThreshold, getStructuringElement(MORPH_RECT, Size(3,
3)));
    dilate(frameThreshold, frameThreshold, getStructuringElement(MORPH_RECT, Size(4,
4)));
    inRange(frameThreshold, 105, 255, frameThreshold);
    return frameThreshold;
}

void resize_image(Mat &input, float alpha=0.15)
{
    cv::resize(input, input, Size(input.cols*alpha, input.rows*alpha));
}

void width_filter(Mat &img, int width_max=22)
{
    int last_point = 0;
    int count_white = 0;
    uchar now_point = 0;
    int middle_step;
    bool black_to_white = false;
    int white_left = 0;
    int white_right = 0;
    bool big_white = false;
    uchar *pixel_l = img.ptr<uchar>(img.rows/4);
    uchar *pixel_c = img.ptr<uchar>(img.rows/2);
    uchar *pixel_h = img.ptr<uchar>(img.rows*3/4);
    for(int c=0; c

```

```

{
    uchar *pixel = img.ptr<uchar>(r);
    for(int c=0; c.cols;c++)
    {
        if(!big_white)
        {
            now_point = *(pixel+c);
            if(now_point>0)
            {
                count_white++;
            }
            if(now_point>0 && last_point==0)
            {
                count_white = 0;
                black_to_white = true;
            }
            else if(now_point==0 && last_point>0)
            {
                black_to_white = false;
            }
            if(black_to_white && (count_white>=width_max))
            {
                c = c-(width_max+1);
                big_white = true;
            }
            last_point = now_point;
        }
        else
            *(pixel+c)=0;
    }
    big_white = false;
}
else
{
    for(int r=0;r.rows;r++)
    {
        uchar *pixel = img.ptr<uchar>(r);
        for(int c=img.cols-1;c>1;c--)
        {
            if(!big_white)
            {
                now_point = *(pixel+c);
                if(now_point>0)
                    count_white++;

```

```

    if(now_point>0 && last_point==0)
    {
        count_white = 0;
        black_to_white = true;
    }
    else if(now_point==0 && last_point>0)
        black_to_white = false;
    if(black_to_white && (count_white>=width_max))
    {
        c = c+(width_max+1);
        big_white = true;
    }
    last_point = now_point;
}
else
    *(pixel+c)=0;
}
big_white = false;
}
}
}

int *Histogram(Mat &img)
{
    //Create histogram with the length of the width of the frame
    vector<int> histogramLane;
    static int LanePosition[2];
    int init_row,end_row;
    Mat ROI_lane;
    Mat frame;
    init_row = img.rows/2;
    end_row = img.rows/2-1;
    img.copyTo(frame);
    for(int i=0;i<img.cols;i++)
    {
        //Region interest
        ROI_lane=frame(Rect(i,init_row ,1,end_row));
        //Normal values
        divide(255,ROI_lane,ROI_lane);
        //add the value
        histogramLane.push_back((int)(sum(ROI_lane)[0]));
    }
    //Find line left
    vector<int>:: iterator LeftPtr;
    LeftPtr = max_element(histogramLane.begin(),histogramLane.begin()+img.cols/2);
    LanePosition[0] = distance(histogramLane.begin(),LeftPtr);
}

```

```

//find line right
vector<int>:: iterator RightPtr;
RightPtr = max_element(histogramLane.begin()+(img.cols/2)+1,histogramLane.end());
LanePosition[1] = distance(histogramLane.begin(),RightPtr);
return LanePosition;
}

void locate_lanes(Mat &img,Mat &out_img)
{
    Mat region_interest;
    left_points.clear();
    right_points.clear();
    int nwindows , margin,minpix;
    int win_y_low = 0, win_y_high = 0, win_xleft_low = 0, win_xleft_high = 0, win_xright_low =
0, win_xright_high = 0;
    int leftx_current,rightx_current;
    int mean_leftx,mean_rightx,count_left,count_right;
    int *locate_histogram;
    bool isMaxLeft;
    bool isMaxRight;
    uchar max_left=0;
    uchar max_right=0;
    uchar now_left_point;
    uchar now_right_point;
    Point max_left_point;
    Point max_right_point;
    nwindows = 12;
    int window_height = img.rows/nwindows;
    locate_histogram = Histogram(img);
    leftx_current = locate_histogram[0];
    rightx_current = locate_histogram[1];
    // Set the width of the windows +/- margin
    margin = 26;
    minpix = 14;
    // Set minimum number of pixels found to recenter window
    for(int window=0;window<nwindows;window++)
    {
        mean_leftx = 0;
        mean_rightx = 0;
        count_left = 0;
        count_right = 0;
        win_y_low = img.rows - (window+1)*window_height;
        win_y_high = img.rows - window*window_height;
        win_xleft_low = leftx_current - margin;
        win_xleft_high = leftx_current + margin;

```

```

win_xright_low = rightx_current - margin;
win_xright_high = rightx_current + margin;
if( win_xleft_low<0)
    win_xleft_low = 0+1;
if(win_xright_high>= img.cols-1)
    win_xright_high = img.cols-1;
if(win_y_high>=img.rows-1)
    win_y_high = img.rows-1;
if(win_y_low<=0)
    win_y_low = 1;
for(int r = win_y_low;r<win_y_high;r++)
{
    uchar *pixel = img.ptr<uchar>(r);
    max_left = 0;
    max_right = 0;
    isMaxLeft = false;
    isMaxRight = false;
    for(int cl = win_xleft_low+1;cl<win_xleft_high;cl++)
    {
        now_left_point = *(pixel+cl);
        if(now_left_point >0 && now_left_point >=max_left)
        {
            max_left_point.x = r;
            max_left_point.y = cl;
            max_left = now_left_point;
            isMaxLeft = true;
        }
    }
    if(isMaxLeft)
    {
        left_points.push_back(max_left_point);
        mean_leftx += max_left_point.y;
        count_left++;
    }
    for(int cr = win_xright_low+1;cr<win_xright_high;cr++)
    {
        now_right_point = *(pixel+cr);
        if(now_right_point >0 && now_right_point >= max_right)
        {
            max_right_point.x = r;
            max_right_point.y = cr;
            max_right = now_right_point;
            isMaxRight = true;
        }
    }
}

```

```

    if(isMaxRight)
    {
        right_points.push_back(max_right_point);
        mean_rightx += max_right_point.y ;
        count_right++;
    }
}
if(count_left>=minpix)
{
    mean_leftx /= count_left;
    leftx_current = mean_leftx;
}
if(count_right>=minpix)
{
    mean_rightx /= count_right;
    rightx_current = mean_rightx;
}
bool regression_left()
{
    if(left_points.size()<133)
        return false;
    long sumX[5] = {0,0,0,0,0};
    long sumY[3] = {0,0,0};
    long pow2 = 0;
    for(auto point=left_points.begin();point!=left_points.end();point++)
    {
        pow2 = (point->x)*(point->x);
        sumX[0]++;
        sumX[1] += (point->x);
        sumX[2] += pow2;
        sumX[3] += pow2*(point->x);
        sumX[4] += pow2*pow2;
        sumY[0] += (point->y);
        sumY[1] += (point->y)*(point->x);
        sumY[2] += (point->y)*pow2;
    }
    solve_system(sumX,sumY,polyleft);
    return true;
}

bool regression_right()
{
    if(right_points.size()<133)

```

```

        return false;
long sumX[5] = {0,0,0,0,0};
long sumY[3] = {0,0,0};
long pow2 = 0;
for(auto point = right_points.begin();point != right_points.end();point++)
{
    pow2 = (point->x)*(point->x);
    sumX[0]++;
    sumX[1] += (point->x);
    sumX[2] += pow2;
    sumX[3] += pow2*(point->x);
    sumX[4] += pow2*pow2;
    sumY[0] += (point->y);
    sumY[1] += (point->y)*(point->x);
    sumY[2] += (point->y)*pow2;
}
solve_system(sumX,sumY,polyright);
return true;
}
void solve_system(long *sX,long *sY,float *x)
{
int n,i,j,k;
n=3;
float a[n][n+1];
//Load items to matrix
a[0][0]=sX[0];
a[0][1]=sX[1];
a[0][2]=sX[2];
a[0][3]=sY[0];

a[1][0]=sX[1];
a[1][1]=sX[2];
a[1][2]=sX[3];
a[1][3]=sY[1];

a[2][0]=sX[2];
a[2][1]=sX[3];
a[2][2]=sX[4];
a[2][3]=sY[2];

//Pivotisation
for (i=0;i<n;i++)
{
    for (k=i+1;k<n;k++)
    {

```

```

        if (abs(a[i][i])<abs(a[k][i]))
    {
        for (j=0;j<=n;j++)
        {
            double temp=a[i][j];
            a[i][j]=a[k][j];
            a[k][j]=temp;
        }
    }
}

//loop to perform the gauss elimination
for (i=0;i<n-1;i++)
{
    for (k=i+1;k<n;k++)
    {
        double t=a[k][i]/a[i][i];
        for (j=0;j<=n;j++)
            a[k][j]=a[k][j]-
t*a[i][j]; //make the elements below the pivot elements equal to zero or eliminate the variables
    }
}
//back-substitution
for (i=n-1;i>=0;i--)
{
    x[i]=a[i][n];
    for (j=i+1;j<n;j++)
    {
        if (j!=i)
            x[i]=x[i]-a[i][j]*x[j];
    }
    x[i]=x[i]/a[i][i];
}
}

void draw_lines(Mat &img)
{
    int columnL;
    int columnR;
    int row;
    float m = 0.0,b = 0.0,c=0.0;
    bool find_line_left;
    bool find_line_right;
    float angle_to_mid_radian;
    find_line_right = regression_right();
}

```

```

find_line_left = regression_left();
right_points.clear();
left_points.clear();
center_cam =(img.cols/2)+1;
for(row = img.rows-1;row>=0;row-=8)
{
    columnR = polyright[0] + polyright[1]*(row)+polyright[2]*(row*row);
    circle(img,cv::Point(columnR,row),cvRound((double)4/ 2), Scalar(255, 0, 0), -1);
    columnL = polyleft[0] + polyleft[1]*(row)+polyleft[2]*(row*row);
    circle(img,cv::Point(columnL,row),cvRound((double)4/ 2), Scalar(255, 0, 0), -1);
    center_lines = (columnR + columnL)/2;
}
distance_center = center_cam - center_lines;
if(distance_center==0)
    angle = 90;
else
{
    angle_to_mid_radian = atan(static_cast<float>(0-(img.rows-
1))/static_cast<float>(center_lines - center_cam));
    angle = static_cast<int>(angle_to_mid_radian * 57.295779);
    if(angle <0 && angle >(0-90))
        angle = (0-1)*(angle);
    else if(angle>0 && angle<90 )
        angle = 180 - angle;
}
line(img,Point(center_cam,(img.rows/4)),Point(center_cam,(img.rows*3/4)),Scalar(0,255,0
),2);
line(img,Point(center_lines,0),Point(center_cam,(img.rows-1)),Scalar(0,0,255),1);
ss.str(" ");
ss.clear();
ss<<"[ANG]: "<<angle;
putText(img, ss.str(), Point2f(2,20), 0,1, Scalar(0,255,255), 1);
}
};

int main(int argc,char **argv)
{
ros::init(argc, argv, "lane_detection");
LineDetection *ic= new LineDetection;
ros::spin();
return 0;
}

```

Apéndice C: crossing_detection.cpp.

```
#include<iostream>
#include<math.h>
#include<string>
#include<ros/ros.h>
#include <std_msgs/String.h>
#include <std_msgs/UInt8.h>
/*Using image transport for publish and subscribing to image in ros,
allows subscriber to compressed image stream*/
#include <image_transport/image_transport.h>
/*Using the headers for CvBridge, to image encodings*/
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
/*Include the headers for opencv image processing and GUI modules */
#include <opencv2/opencv.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
//Library to use time of computer
#include <chrono>
#include <ctime>
using namespace cv;
using namespace std;

//static const std::string OPENCV_WINDOW = "Original";
static const std::string OPENCV_WINDOW = "CROSSING DETECTION";
class CrossDetection
{
private:
    ros::NodeHandle nh_;
    image_transport::ImageTransport it_;
    image_transport::Subscriber image_sub_;
    ros::Publisher cross_pub;
    std_msgs::UInt8 cross_message;
    std::chrono::time_point<std::chrono::system_clock> start,end;
    std::chrono::duration<double> elapsed_seconds;
    stringstream ss,ss_angle;
    /*variable image*/
    Mat frame;
    Mat img_edges;
    Mat img_lines;
    Mat img_perspective;
    vector<Point> cross_points;
    vector<Point> cross_line;
```

```

float polycross[3];
Point2f Source[4];
Point2f Destination[4];
int distance_to_cross;
int center_cam;
int center_lines;
float angle_cross_deg;
public:
    //Constructor
    CrossDetection():it_(nh_){
        // Subscribe to input video feed and publish output video feed
        image_sub_ = it_.subscribe("/jetson_camera/raw",1,&CrossDetection::CrossDetectionCb, this);
        cross_pub = nh_.advertise<std_msgs::UInt8>("/distance_crossing",1000);
    }
    //cv::namedWindow(OPENCV_WINDOW, WINDOW_KEEPATIO);
    Source[0]=Point2f(824*0.2,670*0.2);
    Source[1]=Point2f(1640*0.2,670*0.2);
    Source[2]=Point2f(14*0.2,1200*0.2);
    Source[3]=Point2f(2450*0.2,1200*0.2);
    //init points to desination
    Destination[0]=Point2f(314*0.2,0);
    Destination[1]=Point2f(2150*0.2,0);
    Destination[2]=Point2f(314*0.2,1440*0.2);
    Destination[3]=Point2f(2150*0.2,1440*0.2);
    distance_to_cross = 1000;
    center_cam = 0;
    center_lines = 0;
    angle_cross_deg = 0;
}
//Destructor
~CrossDetection(){
    //cv::destroyWindow(OPENCV_WINDOW);
}
//Callback funtion
void CrossDetectionCb(const sensor_msgs::ImageConstPtr& msg){
    set_start_time();
    cv_bridge::CvImagePtr cv_ptr;
    try{
        cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::BGR8);
        copyImageRGB(cv_ptr->image,frame);
    }
    catch (cv_bridge::Exception& e){
        ROS_ERROR("cv_bridge exception: %s", e.what());
    }
}

```

```

        return;
    }
    Perspective(frame);
    cv::rotate(frame,frame,ROTATE_90_COUNTERCLOCKWISE);
    img_edges = Threshold(frame);
    locate_lanes(img_edges,frame);
    draw_lines(frame);
//    resizeWindow(OPENCV_WINDOW,frame.cols,frame.rows);
//    cv::imshow(OPENCV_WINDOW,frame);
//    cv::waitKey(5);
// Output modified video stream
if(angle_cross_deg>=80.0 && angle_cross_deg<=100.0)
{
    cross_message.data = distance_to_cross;
    cross_pub.publish(cross_message);
}
else
{
    cross_message.data = img_perspective.cols-1;
    cross_pub.publish(cross_message);
}
set_end_time();
ROS_INFO("[FPS]: %i ",FPS_subscriber());
}
void set_start_time()
{
    start = std::chrono::system_clock::now();
}
void set_end_time()
{
    end = std::chrono::system_clock::now();
}
int FPS_subscriber()
{
    elapsed_seconds = end-start;
    float t = elapsed_seconds.count();
    return 1/t;
}
//Method to change image from BGR to RGB
void copyImageRGB(Mat & image_BGR,Mat &image_RGB)
{
    cvtColor(image_BGR, image_RGB, COLOR_BGR2RGB);
}
void Perspective(Mat &frame)
{

```

```

//Mat output;
Mat matrixPerspective( 2, 4, CV_32FC1 );
matrixPerspective = Mat::zeros( frame.rows, frame.cols, frame.type());
matrixPerspective = getPerspectiveTransform(Source, Destination);

warpPerspective(frame,frame,matrixPerspective,Size(frame.cols,frame.rows));
}

Mat Threshold(Mat frame)
{
    Mat frameThreshold;
    frame.convertTo(frame, -1, 1, -15);
    cvtColor(frame,frameThreshold,COLOR_BGR2GRAY);
    medianBlur(frameThreshold,frameThreshold,5);
    erode(frameThreshold,frameThreshold, getStructuringElement(MORPH_RECT, Size(3,3)));
    dilate(frameThreshold,frameThreshold, getStructuringElement(MORPH_RECT, Size(4,4)));
    inRange(frameThreshold,105,255,frameThreshold);
    return frameThreshold;
}
Mat resize_image(Mat input,float alpha=0.5)
{
    Mat output;
    cv::resize(input,output,Size(input.cols*alpha,input.rows*alpha));
;
    return output;
}
void width_filter(Mat &img,int width_max=30)
{
    int last_point = 0;
    int count_white = 0;
    uchar now_point = 0;
    int middle_step;
    bool black_to_white = false;
    int white_left = 0;
    int white_right = 0;
    bool big_white = false;
    uchar *pixel_l = img.ptr<uchar>(img.rows/4);
    uchar *pixel_c = img.ptr<uchar>(img.rows/2);
    uchar *pixel_h = img.ptr<uchar>(img.rows*3/4);
    for(int c =0; c<img.cols;c++)
    {

```

```

        if(*((pixel_l+c)>0 || *(pixel_c+c)>0||*(pixel_h+c)>0)
    {
        if(c<img.cols/2)
            white_left++;
        else if(c>=img.cols/2)
            white_right++;
    }
}
if(white_right>white_left)
{
    for(int r=0;r<img.rows;r++)
    {
        uchar *pixel = img.ptr<uchar>(r);
        for(int c =0; c<img.cols;c++)
        {
            if(!big_white)
            {
                now_point = *(pixel+c);
                if(now_point>0)
                    count_white++;
                if(now_point>0 && last_point==0)
                {
                    count_white = 0;
                    black_to_white = true;
                }
                else if(now_point==0 && last_point>0)
                    black_to_white = false;
                if(black_to_white && (count_white>=width_max))
                {
                    c = c-(width_max+1);
                    big_white = true;
                }
                last_point = now_point;
            }
            else
                *(pixel+c)=0;
        }
        big_white = false;
    }
}
else
{
    for(int r=0;r<img.rows;r++)
    {
        uchar *pixel = img.ptr<uchar>(r);

```

```

        for(int c =img.cols-1;c>1;c--)
        {
            if(!big_white)
            {
                now_point = *(pixel+c);
                if(now_point>0)
                    count_white++;
                if(now_point>0 && last_point==0)
                {
                    count_white = 0;
                    black_to_white = true;
                }
                else if(now_point==0 && last_point>0)
                    black_to_white = false;
                if(black_to_white && (count_white>=width_max))
                {
                    c = c+(width_max+1);
                    big_white = true;
                }
                last_point = now_point;
            }
            else
                *(pixel+c)=0;
        }
        big_white = false;
    }
}
int *Histogram(Mat &img)
{
    //Create histogram with the length of the width of the frame
    vector<int> histogramLane;
    static int LanePosition[2];
    int init_row,end_row;
    Mat ROI_lane;
    Mat frame;
    init_row = img.rows/2;
    end_row = img.rows/2-1;
    img.copyTo(frame);
    for(int i=0;i<img.cols;i++)
    {
        //Region interest
        ROI_lane=frame(Rect(i,init_row ,1,end_row));
        //Normal values
        divide(255,ROI_lane,ROI_lane);
    }
}

```

```

        //add the value
        histogramLane.push_back((int)(sum(ROILane)[0]));
    }
    //Find line left
    vector<int>:: iterator LeftPtr;
    LeftPtr = max_element(histogramLane.begin(),histogramLane.begin(
)+img.cols/2);
    LanePosition[0] = distance(histogramLane.begin(),LeftPtr);
    //find line right
    vector<int>:: iterator RightPtr;
    RightPtr = max_element(histogramLane.begin()+(img.cols/2)+1,hist
ogramLane.end());
    LanePosition[1] = distance(histogramLane.begin(),RightPtr);
    return LanePosition;
}

void locate_lanes(Mat &img,Mat &out_img)
{
    Mat region_interest;
    cross_points.clear();
    int nwindows , margin,minpix;
    int win_y_low = 0, win_y_high = 0, win_xcross_low = 0, win_xcross_hi
gh = 0;
    int crossx_current;
    int mean_crossx,count_cross;
    int *locate_histogram;
    bool isMaxCross;
    uchar max_cross=0;
    uchar now_cross_point;
    Point max_cross_point;
    nwindows = 12;
    int window_height = img.rows/nwindows;
    locate_histogram = Histogram(img);
    crosstx_current = locate_histogram[0];
    // Set the width of the windows +/- margin
    margin = 40;
    minpix = 14;
    // Set minimum number of pixels found to recenter window
    for(int window=0;window<nwindows;window++)
    {
        mean_crossx = 0;
        count_cross = 0;
        win_y_low = img.rows - (window+1)*window_height;
        win_y_high = img.rows - window*window_height;
        win_xcross_low = crossx_current - margin;

```

```

    win_xcross_high = crossx_current + margin;
    if( win_xcross_low<0)
        win_xcross_low = 0+1;
    if(win_xcross_high>= img.cols-1)
        win_xcross_high = img.cols-1;
    if(win_y_high>=img.rows-1)
        win_y_high = img.rows-1;
    if(win_y_low<=0)
        win_y_low = 1;
    for(int r = win_y_low;r<win_y_high;r++)
    {
        max_cross = img.at<uchar>(r,win_xcross_low);
        isMaxCross = false;
        for(int cl = win_xcross_low+1;cl<win_xcross_high;cl++)
        {
            now_cross_point = img.at<uchar>(r,cl);
            if(now_cross_point >0 && now_cross_point >=max_cross)
            {
                max_cross_point.x = r;
                max_cross_point.y = cl;
                max_cross = now_cross_point;
                isMaxCross = true;
            }
        }
        if(isMaxCross)
        {
            cross_points.push_back(max_cross_point);
            mean_crossx += max_cross_point.y;
            count_cross++;
        }
    }
    if(count_cross>=minpix)
    {
        mean_crossx /= count_cross;
        crossx_current = mean_crossx;
    }
    rectangle(out_img,cv::Point(win_xcross_low,win_y_low),cv::Point(win_xcross_high,win_y_high),cv::Scalar(0,255,0));
}
bool regression_cross()
{
    if(cross_points.size()<100)
        return false;
    long sumX[3] = {0,0,0};

```

```

long sumY[2] = {0,0};
long pow2 = 0;
for(auto point=cross_points.begin();point!=cross_points.end();point+
+)
{
    pow2 = (point->x)*(point->x);
    sumX[0]++;
    sumX[1]+= (point->x);
    sumX[2]+= pow2;
    sumY[0]+= (point->y);
    sumY[1]+= (point->y)*(point->x);
}
solve_system(sumX,sumY,polycross);
return true;
}
void solve_system(long *sX,long *sY,float *x
{
    int n,i,j,k;
    n=2;
    float a[n][n+1];
    //Enter the elements of the augmented-matrix row-wise
    a[0][0]=sX[0];
    a[0][1]=sX[1];
    a[0][2]=sY[0];

    a[1][0]=sX[1];
    a[1][1]=sX[2];
    a[1][2]=sY[1];

    //Pivotisation
    for (i=0;i<n;i++)
    {
        for (k=i+1;k<n;k++)
        {
            if (abs(a[i][i])<abs(a[k][i]))
            {
                for (j=0;j<=n;j++)
                {
                    double temp=a[i][j];
                    a[i][j]=a[k][j];
                    a[k][j]=temp;
                }
            }
        }
    }
}

```

```

//loop to perform the gauss elimination
for (i=0;i<n-1;i++)
{
    for (k=i+1;k<n;k++)
    {
        double t=a[k][i]/a[i][i];
        for (j=0;j<=n;j++)
            a[k][j]=a[k][j]-
t*a[i][j]; //make the elements below the pivot elements equal to zero or
eliminate the variables
    }
}
//back-substitution
for (i=n-1;i>=0;i--)
{
    x[i]=a[i][n];
    for (j=i+1;j<n;j++)
    {
        if (j!=i)
            x[i]=x[i]-a[i][j]*x[j];
    }
    x[i]=x[i]/a[i][i];
}
void draw_lines(Mat &img)
{
    int column;
    int row;
    Point half_line,half_start;
    center_cam =(img.cols/2)+1;
    center_lines =0;
    float angle_cross_radian;
    cross_line.clear();
    if(regression_cross())
    {
        for(row=(img.rows/4);row<(img.rows*3/4)-1;row+=6)
        {
            column = polycross[0] + polycross[1]*(row);
            cross_line.push_back(cv::Point(column,row));
            circle(img,cv::Point(column,row),cvRound((double)4/ 2), Scalar(255,0,0), -3);
        }
        cross_line.x = ((cross_line.end()-1)->x +cross_line.begin()-
>x)/2;
        cross_line.y = img.rows/2;
    }
}

```

```

        cross_start.x = (img.cols-1);
        cross_start.y = img.rows/2;
        line(img,half_line,half_start,Scalar(0,0,255),4);
        distance_to_cross = half_start.x - half_line.x;
        angle_cross_radian = atan(static_cast<float>((cross_line.end()-
1)->y - cross_line.begin()->y)/static_cast<float>((cross_line.end()-1)-
>x - cross_line.begin()->x));
        angle_cross_deg = static_cast<int>(angle_cross_radian * 180.0 /
3.14159265);
        if(angle_cross_deg<0 && angle_cross_deg>(0-90))
            angle_cross_deg = (0-1)*(angle_cross_deg);
        else if(angle_cross_deg>0 && angle_cross_deg<90 )
            angle_cross_deg = 180 - angle_cross_deg;
    }
    else
        angle_cross_deg = 0;
    ss.str(" ");
    ss.clear();
    ss<<"[DIST]: "<<distance_to_cross;
    putText(img, ss.str(), Point2f(2,20), 0,1, Scalar(0,255,255), 1);
}
};

int main(int argc,char **argv)
{
    ros::init(argc, argv, "crossing_detection");
    CrossDetection *ic= new CrossDetection;
    ros::spin();
    return 0;
}

```