



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA

  
ETSI Aeroespacial y Diseño Industrial

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial  
y Diseño Industrial

SISTEMA DE VISIÓN INTELIGENTE PARA EL GUIADO  
DE UN VEHÍCULO AUTÓNOMO EN UN ENTORNO  
URBANO

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Valencia Torres, Mario

Tutor/a: Rodas Jordá, Ángel

CURSO ACADÉMICO: 2023/2024



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA

  
ETSI Aeroespacial y Diseño Industrial

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial  
y Diseño Industrial

SISTEMA DE VISIÓN INTELIGENTE PARA EL GUIADO  
DE UN VEHÍCULO AUTÓNOMO EN UN ENTORNO  
URBANO

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Valencia Torres, Mario

Tutor/a: Rodas Jordá, Ángel

CURSO ACADÉMICO: 2023/2024



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



# Dedicatoria

---

A los fracasos, a las causas perdidas y a los que se fueron para no irse nunca.



# Agradecimientos

---

Primeramente, agradecer encarecidamente a mi tutor la constante ayuda ofrecida durante toda la realización del proyecto, desde la confección de este hasta el último detalle antes de firmar este documento.

Por supuesto, agradecer a mis padres las incalculables lecciones y las numerosas ayudas que me han hecho ser lo que soy hoy en día y lo que seré en el futuro.

A los que se fueron pero escriben estas palabras, mi vida es solo la representación de todo lo que sois.

A mis amigos, dese por aludido quien así lo desee, por todos los momentos compartidos y por los que nos esperan, sin vosotros nada tiene sentido, hacéis de la vida un lugar precioso. A S.A.G, desde que tenemos memoria juntos hasta que nada quede, nada de lo que se pueda leer aquí dirá nunca más que nuestras miradas.

Finalmente, a mi hermano, la persona más importante en mi vida, las estrellas a la noche y el fuego a la oscuridad, mi guía, mi protector, mi único ejemplo a seguir, no existe tiempo suficiente para agradecer lo que aportas cada segundo.

## Resumen

---

Este Trabajo de Fin de Grado tiene como objetivo la implementación, mediante técnicas de visión artificial, redes neuronales convolucionales y técnicas de planificación y seguimiento de trayectoria, el control de un vehículo Ackerman para desplazarse de manera autónoma por un entorno urbano. Para poder realizar dicho trabajo se utilizarán dos programas, Coppelia Sim y Matlab, para simular el entorno e implementar el algoritmo de control respectivamente.

Para ello, se incorporará al vehículo una cámara frontal para captar la información necesaria del entorno por el que circula el vehículo. Una vez esta información ha sido captada, se transmitirá a Matlab para obtener las características más importantes de esta y actuar en consecuencia.

A continuación, y con el fin de elegir la opción más eficiente, se han utilizado distintas técnicas de identificación para poder compararlas entre sí y elegir posteriormente la que mejores resultados de, no solo en la simulación, sino de cara a una futura escalabilidad en entornos más complejos. Para la identificación de las señales, se utilizarán técnicas de *machine learning* y más en particular redes neuronales convolucionales ya que son idóneas para problemas de este tipo.

Por otra parte, se creará un sistema complejo de visión inteligente en el cual se detecte con total claridad cada componente del escenario, cielo, edificios, arcén, césped... para poder así segmentar el carril que debe seguir el vehículo en todo momento. Sin embargo, no solo será suficiente con detectar el carril sino que se creará un algoritmo inteligente mediante el cual, teniendo en cuenta las líneas del carril, se controle un vehículo de tipo Ackerman.

Cabe destacar la adaptabilidad de este algoritmo para lograr no solo incorporarse a un carril o seguir este, sino tomar posibles curvas que aparezcan a lo largo del recorrido independientemente de su sentido o ángulo de giro. Además, se han utilizado técnicas de planificación de rutas, y seguimiento de ellas, para, utilizando el GPS del vehículo calcular la ruta más corta para completar el trayecto deseado en el menor tiempo posible.

Para finalizar, se han puesto en común los distintos algoritmos creados tanto para cumplir lo anteriormente mencionado, como para la conexión y la transmisión de imágenes en tiempo real entre ambos programas.

El desarrollo de vehículos autónomos es uno de los campos de investigación más activos en la actualidad y estos métodos de simulación permiten trabajar de manera mucho más barata y rápida en estos aspectos. Además, este modelo de simulación también cuenta con una gran utilidad a la hora de divulgar las últimas técnicas de visión artificial en centros educativos sin la necesidad, ni los costes que conllevaría recrear este escenario y dotar al robot de toda la tecnología necesaria para su correcta implementación.

**Palabras clave:** CoppeliaSim; Matlab; Visión Artificial; Redes Neuronales Convolucionales (CNN); Deep Learning; Planificación de trayectorias; Ackerman.

# Summary

---

The objective of this Final Degree Project is to implement, by means of cutting-edge artificial vision techniques, convolutional neural networks and route planning and tracking techniques, the control of a Ackerman vehicle to move autonomously through an urban environment. Two programs, Coppelia Sim and Matlab, will be used to simulate the environment and implement the control algorithm, respectively.

For this purpose, a front camera will be incorporated into the vehicle to capture the necessary information about the environment in which the vehicle circulates. Once this information has been captured, it will be transmitted to Matlab to obtain the most essential characteristics of the vehicle and act accordingly.

Furthermore, in order to choose the most efficient option, different identification techniques were used to compare them and then choose the one that gives the best results not only in the simulation but also for future scalability in more complex environments. For the identification of the signals, machine learning techniques will be used and more particularly convolutional neural networks, since they are ideal for problems of this type.

On the other hand, a sophisticated and complex intelligent vision system will be created in which each component of the scenario, sky, buildings, shoulder, grass... will be detected with total clarity in order to be able to segment the lane that the vehicle must follow at all times. However, it will not only be enough to detect the lane, but an intelligent algorithm will be created by means of which, taking into account the lane lines, a Ackerman-type vehicle will be controlled.

It should be noted the adaptability of this algorithm to not only join a lane or follow it but also to take possible curves that appear throughout the simulation regardless of their direction or angle of rotation. In addition, route planning and tracking techniques have been used to calculate the shortest route to complete the desired route in the shortest possible time using the vehicle's GPS.

Finally, the different algorithms created to accomplish the above-mentioned, as well as for connecting and transmitting images in real time between the two programs, have been put in common.

The development of autonomous vehicles is one of the most active research fields nowadays and these simulation methods allow working much cheaper and faster in these aspects. In addition, this simulation model also has a great utility when it comes to disseminating the latest computer vision techniques in educational centers without the need or the costs that would entail recreating this scenario and providing the robot with all the necessary technology for its proper implementation.

**Keywords:** CoppeliaSim; Matlab; Computer Vision; Convolutional Neural Networks (CNN); Deep Learning; Path Planning; Ackerman.



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



# UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

---

## SISTEMA DE VISIÓN INTELIGENTE PARA EL GUIADO DE UN VEHÍCULO AUTÓNOMO EN UN ENTORNO URBANO

TRABAJO FINAL DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL  
Y AUTOMÁTICA

Documento

Índice



# Índice general

---

|                            |     |
|----------------------------|-----|
| Resumen.....               | 3   |
| Índice general.....        | 6   |
| Memoria.....               | 7   |
| Bibliografía.....          | 131 |
| Anexos.....                | 138 |
| Pliego de condiciones..... | 165 |
| Presupuesto.....           | 170 |





UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



# UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

---

## SISTEMA DE VISIÓN INTELIGENTE PARA EL GUIADO DE UN VEHÍCULO AUTÓNOMO EN UN ENTORNO URBANO

TRABAJO FINAL DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL  
Y AUTOMÁTICA

Documento

Memoria



# Índice de Contenidos

---

|  |    |
|--|----|
| 1. Introducción, motivación y objetivos.....               | 18 |
| 1.1. Introducción .....                                    | 18 |
| 1.2. Motivación .....                                      | 18 |
| 1.2.1. Seguridad vial.....                                 | 19 |
| 1.2.2. Robótica educativa .....                            | 19 |
| 1.3. Objetivos .....                                       | 20 |
| 1.4. Esquema general .....                                 | 20 |
| 2. Estado del Arte .....                                   | 20 |
| 2.1. Investigaciones públicas previas .....                | 26 |
| 2.2. Algoritmos de planificación de ruta .....             | 27 |
| 2.3. Algoritmos de visión por computador .....             | 28 |
| 3. Especificaciones de diseño y conocimientos previos..... | 29 |
| 3.1. Coppelia Sim.....                                     | 29 |
| 3.1.1. Interfaz de usuario.....                            | 31 |
| 3.1.2. Modelos y objetos .....                             | 33 |
| 3.1.3. Escena CoppeliaSim.....                             | 34 |
| 3.2. Conexionado Matlab-Coppelia .....                     | 35 |
| 3.3. Matlab .....  | 36 |
| 3.3.1. Deep Network Designer .....                         | 38 |
| 3.3.2. Color Thresholder .....                             | 41 |
| 3.4. Redes neuronales convolucionales (CNN).....           | 44 |
| 3.5. Morfología matemática .....                           | 46 |
| 3.6. GPS y planificación de la trayectoria.....            | 47 |
| 4. Detección de señales .....                              | 47 |
| 4.1. Creación de la Data base .....                        | 48 |
| 4.1.1. Escena Coppelia.....                                | 48 |
| 4.1.2. Obtención de las imágenes.....                      | 49 |
| 4.2. Preentrenamiento.....                                 | 51 |
| 4.2.1. Labeling.....                                       | 52 |
| 4.2.2. Resizing.....                                       | 52 |
| 4.2.3. Splitting.....                                      | 53 |
| 4.3. Soluciones propuestas .....                           | 54 |
| 4.3.1. SqueezeNet.....                                     | 54 |



|   |     |
|---|-----|
| 4.3.1.1. Entrenamiento .....                                      | 55  |
| 4.3.1.2. Resultados (Test) .....                                  | 58  |
| 4.3.2. Resnet50 .....   | 61  |
| 4.3.2.1. Entrenamiento .....                                      | 61  |
| 4.3.2.2. Resultados .....   | 64  |
| 4.3.3. YOLOv2 .....   | 65  |
| 4.3.3.1. Entrenamiento .....                                      | 66  |
| 4.3.3.2. Resultados .....   | 68  |
| 4.4. Selección modelo .....                                       | 69  |
| 4.5. Implementación del logaritmo de detección .....              | 70  |
| 4.6. Respuesta a las señales .....                                | 73  |
| 4.6.1. Velocidad estándar .....                                   | 74  |
| 4.6.2. Frenado .....  | 74  |
| 4.6.3. Parada .....   | 75  |
| 4.6.3.1. Señales de prohibido y callejón .....                    | 76  |
| 4.6.3.2. Señal de Stop .....                                      | 76  |
| 4.6.3.3. Semáforo .....   | 78  |
| 5. Detección del carril y cálculos de distancia .....             | 80  |
| 5.1. Detección del carril .....                                   | 80  |
| 5.1.1. Análisis del entorno .....                                 | 81  |
| 5.1.2. Eliminación del ruido .....                                | 84  |
| 5.1.3. Trabajo sobre la línea divisoria .....                     | 87  |
| 5.1.4. Obtención de los bordes laterales .....                    | 89  |
| 5.1.5. Adaptación para curvas .....                               | 90  |
| 5.1.6. Obtención de los ángulos y del centroide derecho .....     | 90  |
| 5.1.7. Control del vehículo .....                                 | 92  |
| 5.1.7.1. Obtención de los ángulos de giro .....                   | 93  |
| 5.1.7.2. Transmisión de la información .....                      | 95  |
| 5.1.8. Pruebas de estrés .....                                    | 95  |
| 5.1.8.1. Entrada en el carril .....                               | 96  |
| 5.1.8.2. Funcionamiento en curvas .....                           | 98  |
| 5.2. Detección de la barra de la señalización .....               | 100 |
| 6. Algoritmo de planificación y seguimiento de trayectorias ..... | 103 |
| 6.1. Obtención de las rutas .....                                 | 104 |
| 6.2. Clasificación de las rutas .....                             | 106 |



|   |     |
|---|-----|
| 6.3. Obtención de la información .....                        | 108 |
| 6.3.1. Ruta de origen .....                                   | 108 |
| 6.3.2. Ruta de destino .....                                  | 109 |
| 6.3.3. Cálculo de la trayectoria más corta .....              | 110 |
| 6.4. Seguimiento de la trayectoria .....                      | 111 |
| 6.5. Transmisión de los datos .....                           | 115 |
| 6.6. Pruebas del algoritmo .....                              | 115 |
| 6.6.1. Reconducción de la trayectoria .....                   | 115 |
| 6.6.2. Paso por curva .....                                   | 117 |
| 7. Implementación final .....                                 | 119 |
| 7.1. Inicialización .....                                     | 121 |
| 7.1.1. Adición de las funciones auxiliares .....              | 121 |
| 7.1.2. Inicialización de las variables .....                  | 121 |
| 7.1.3. Inicialización de la simulación .....                  | 122 |
| 7.1.4. Inicialización del seguimiento de la trayectoria ..... | 122 |
| 7.2. Seguimiento de la trayectoria .....                      | 123 |
| 8. Estudio de aplicabilidad .....                             | 127 |
| 8.1. Aplicación del modelo .....                              | 127 |
| 8.2. Trabajos futuros .....                                   | 128 |
| 9. Conclusión .....   | 129 |

# Índice de Figuras

---

|   |    |
|---|----|
| Figura 1. Esquema general del algoritmo.....  | 20 |
| Figura 2. Ilustración sobre la captación de información sobre el entorno [2].....       | 21 |
| Figura 3. Las calles del futuro según la maqueta Futurama (1939) .....                  | 22 |
| Figura 4. Evolución de las víctima mortales en accidentes de tráfico en España [6]..... | 23 |
| Figura 5. Niveles de conducción autónoma .....  | 24 |
| Figura 6. Interior de la furgoneta de Dickmanns.....                                    | 25 |
| Figura 7. Vehículo Waymo en funcionamiento [12].....                                    | 25 |
| Figura 8. Grafo ejemplificativo [25] .....  | 27 |
| Figura 9. Tabla comparativa de los 6 métodos de control .....                           | 30 |
| Figura 10. Ventana de consola de CoppeliaSim .....                                      | 31 |
| Figura 11. Ventana de aplicación de CoppeliaSim .....                                   | 32 |
| Figura 12. Ventana de aplicación dividida en partes .....                               | 32 |
| Figura 13. Categoría de objetos de CoppeliaSim .....                                    | 33 |
| Figura 14. Escena CoppeliaSim .....   | 34 |
| Figura 15. Señales escena Coppelia .....  | 35 |
| Figura 16. Modelo del vehículo autónomo .....   | 35 |
| Figura 17. Archivos necesarios para la interconexión.....                               | 36 |
| Figura 18. Configuración conexionado en Coppelia.....                                   | 36 |
| Figura 19. Ofertas Matlab para estudiantes [34] .....                                   | 37 |
| Figura 20. Esquema arquitectura del directorio del código del proyecto.....             | 37 |
| Figura 21. Conjunto de aplicaciones disponibles en Matlab.....                          | 38 |
| Figura 22. Redes preentrenadas disponibles en Deep Network Designer.....                | 39 |
| Figura 23. Gráfica comparativa redes Deep Network Designer.....                         | 40 |
| Figura 24. Ventana principal ejemplificativa .....                                      | 41 |
| Figura 25. Espacio de colores disponibles [39].....                                     | 42 |
| Figura 26. Ventana inicial para la segmentación por color .....                         | 43 |
| Figura 27. Segmentación según el espacio de color RGB.....                              | 43 |
| Figura 28. Explicación aplicación kernel convolucional [40].....                        | 44 |
| Figura 29. Aplicación de un filtro convolucional.....                                   | 44 |
| Figura 30. Obtención del mapa de características [41] .....                             | 45 |
| Figura 31. Red neuronal convolucional [40].....   | 45 |
| Figura 32. Arquitectura de una CNN [42].....  | 46 |
| Figura 33. Aplicación sobre una imagen en escala de grises [43].....                    | 46 |

|   |    |
|---|----|
| Figura 34. Agrupación de las rutas en Coppelia.....                               | 47 |
| Figura 35. Esquema organizativo de las rutas .....                                | 47 |
| Figura 36. Planeo cenital de la escena entrenamiento .....                        | 49 |
| Figura 37. Código para iniciar la simulación .....                                | 49 |
| Figura 38. Configuración características simulación .....                         | 50 |
| Figura 39. Función getImage .....   | 50 |
| Figura 40. Código principal para hacer la Data base .....                         | 51 |
| Figura 41. Etiquetado automático de las imágenes .....                            | 52 |
| Figura 42. Data base ordenada y etiquetada.....                                   | 52 |
| Figura 43. Comando imresize .....   | 53 |
| Figura 44. Dirección carpetas para el Split .....                                 | 53 |
| Figura 45. Obtención de los índices de las imágenes.....                          | 53 |
| Figura 46. Bucles para el Split de las imágenes de la base de datos .....         | 54 |
| Figura 47. Imagen entrenamiento SqueezeNet.....                                   | 55 |
| Figura 48. Arquitectura red SqueezeNet .....                                      | 55 |
| Figura 49. Opciones para importar la base de datos.....                           | 56 |
| Figura 50. Training Data para SqueezeNet .....                                    | 56 |
| Figura 51. Opciones entrenamiento SqueezeNet .....                                | 57 |
| Figura 52. Resultados entrenamiento SqueezeNet.....                               | 57 |
| Figura 53. Dirección de las imágenes del test.....                                | 58 |
| Figura 54. Dirección carpeta para los fallos.....                                 | 58 |
| Figura 55. Función selección de colores .....                                     | 59 |
| Figura 56. Gráfica resultados SqueezeNet .....                                    | 59 |
| Figura 57. Resultados SqueezeNet.....   | 60 |
| Figura 58. Escritura en la imagen.....  | 60 |
| Figura 59. Fotograma ejemplificativo del video creado por el script de test ..... | 60 |
| Figura 60. Arquitectura modificada Resnet50.....                                  | 61 |
| Figura 61. Data de entrenamiento Resnet50 .....                                   | 62 |
| Figura 62. Opciones entrenamiento Resnet50.....                                   | 63 |
| Figura 63. Gráfica del entrenamiento de Resnet50 .....                            | 63 |
| Figura 64. Gráfica aciertos y fallos red Resnet50 .....                           | 64 |
| Figura 65. Resultados Resnet50 sobre las imágenes de test.....                    | 64 |
| Figura 66. Frame ejemplificativo video Resnet50 .....                             | 65 |
| Figura 67. Detección en una imagen por una red de arquitectura YOLO [46] .....    | 66 |
| Figura 68. Arquitectura red YOLOv2 [47].....                                      | 66 |

|  |    |
|--|----|
| Figura 69. Capas del entrenamiento de la red YOLO .....                                  | 67 |
| Figura 70. Reducción del overfitting.....  | 67 |
| Figura 71. Opciones de entrenamiento de la red YOLO .....                                | 68 |
| Figura 72. Resultados entrenamiento red YOLOv2 .....                                     | 68 |
| Figura 73. Gráfica aciertos y fallos red YOLOv2 .....                                    | 69 |
| Figura 74. Resultados precisión red YOLOv2 .....   | 69 |
| Figura 75. Esquema cabecera de la función para la detección de señales.....              | 70 |
| Figura 76. Análisis de la imagen por la CNN.....   | 71 |
| Figura 77. Esquema lógico para la detección.....   | 71 |
| Figura 78. Clasificación de la imagen por la red neuronal.....                           | 72 |
| Figura 79. Secuencia lógica para el procesamiento de la detección.....                   | 72 |
| Figura 80. Implementación de la última parte del algoritmo de detección de señales ..... | 72 |
| Figura 81. Esquema variable actuador para las señales .....                              | 73 |
| Figura 82. Árbol de decisión según la detección realizada .....                          | 74 |
| Figura 83. Implementación sentido único, 30 y nada.....                                  | 74 |
| Figura 84. Implementación parking y rotonda .....  | 75 |
| Figura 85. Función auxiliar para el frenado .....  | 75 |
| Figura 86. Implementación prohibido, stop, semáforo y callejón.....                      | 75 |
| Figura 87. Implementación función frenado_to_0.....                                      | 76 |
| Figura 88. Esquema cabecera detección en stop .....                                      | 76 |
| Figura. 89. Esquema lógico detencion_stop.....   | 77 |
| Figura 90. Implementación para la parada en el stop.....                                 | 77 |
| Figura 91. Segmentación para semáforo en rojo.....                                       | 78 |
| Figura 92. Segmentación para semáforo en ámbar .....                                     | 78 |
| Figura 93. Segmentación para semáforo en verde .....                                     | 79 |
| Figura 94. Detección del color del semáforo .....  | 79 |
| Figura 95. Código de la actuación ante semáforos.....                                    | 80 |
| Figura 96. Esquema cabeceras de la función detección_carril_mvalor .....                 | 81 |
| Figura 97. Segmentación por color del césped.....  | 82 |
| Figura 98. Segmentación por color del arcén.....   | 82 |
| Figura 99. Segmentación por color de la línea divisoria.....                             | 83 |
| Figura 100. Segmentación por color del cielo.....  | 83 |
| Figura 101. Código para la clasificación del entorno.....                                | 84 |
| Figura 102. Clasificación final y conjunta .....   | 84 |
| Figura 103. Código para la eliminación de los bordes laterales.....                      | 85 |

|  |     |
|--|-----|
| Figura 104. Función para obtener los dos objetos principales .....                             | 86  |
| Figura 105. Imagen con los objetos principales .....   | 86  |
| Figura 106. Aplicación del comando imfill.....   | 87  |
| Figura 107. Código para obtener las coordenadas del punto superior de la línea divisoria ..... | 88  |
| Figura 108. Recorte del carril izquierdo.....  | 88  |
| Figura 109. Imagen resultante tras recortar el carril izquierdo .....                          | 89  |
| Figura 110. Implementación del algoritmo de Canny .....  | 89  |
| Figura 111. Implementación del algoritmo de Canny .....  | 89  |
| Figura 112. Código para la adaptación a curvas.....  | 90  |
| Figura 113. Bordes laterales una vez aplicada la adaptación a las curvas.....                  | 90  |
| Figura 114. Identificación y clasificación de las líneas.....                                  | 91  |
| Figura 115. Código para la identificación de los ángulos del carril .....                      | 92  |
| Figura 116. Obtención del centroide del carril derecho .....                                   | 92  |
| Figura 117. Esquema funcionamiento control Ackerman [55].....                                  | 93  |
| Figura 118. Esquema función para los ángulos de giro.....                                      | 94  |
| Figura 119. Implementación para los ángulos de giro .....                                      | 95  |
| Figura 120. Implementación para transmitir la información.....                                 | 95  |
| Figura 121. Modificación para las pruebas del carril .....                                     | 96  |
| Figura 122. Inicio del test.....   | 96  |
| Figura 123. Reconducción de la situación.....  | 97  |
| Figura 124. Dirección centrada y restablecimiento del seguimiento de la trayectoria.....       | 97  |
| Figura 125. Escena para realizar el test en curvas.....  | 98  |
| Figura 126. Test de detección del carril en una curva de derechas .....                        | 99  |
| Figura 127. Test de detección del carril en una curva de izquierdas .....                      | 99  |
| Figura 128. Esquema de la cabecera de la función detectBar .....                               | 100 |
| Figura 129. Interfaz del Color Thresholder para la detección de la barra .....                 | 101 |
| Figura 130. Aplicación de la morfología matemática en la detección de la barra.....            | 101 |
| Figura 131. Comparación para la morfología matemática en detectBar .....                       | 101 |
| Figura 132. Código para detectar la barra más cercana .....                                    | 102 |
| Figura 133. Parámetros para el cálculo de la distancia .....                                   | 103 |
| Figura 134. Esquema del pinhole [56] .....   | 103 |
| Figura 135. Estructura de los paths del entorno .....  | 104 |
| Figura 136. Código para la agrupación de los paths en una variable .....                       | 104 |
| Figura 137. Función create_path_list.....  | 105 |
| Figura 138. Función createSimPath .....  | 105 |





|  |     |
|--|-----|
| Figura 139. Función createSignList .....   | 105 |
| Figura 140. Función createSign .....   | 106 |
| Figura 141. Esquema cálculo de distancia [57].....   | 106 |
| Figura 142. Función auxiliar para el cálculo de la distancia .....                         | 106 |
| Figura 143. Función auxiliar para la clasificación de las rutas .....                      | 107 |
| Figura 144. Código para mostrar los posibles caminos.....                                  | 107 |
| Figura 145. Grafos del conexionado de las rutas del entorno .....                          | 108 |
| Figura 146. Función auxiliar para la obtención del punto de partida.....                   | 109 |
| Figura 147. Obtención de la ruta destino .....   | 110 |
| Figura 148. Función auxiliar para la obtención de la información de origen / destino ..... | 110 |
| Figura 149. Implementación de la función getInfo .....                                     | 111 |
| Figura 150. Imagen ejemplificativa de la trayectoria.....                                  | 111 |
| Figura 151. Esquema de la cabecera de la función auxiliar wheelControlFromRoute .....      | 112 |
| Figura 152. Obtención del punto más cercano .....  | 112 |
| Figura 153. Pruebas para la selección del punto referencia .....                           | 113 |
| Figura 154. Seguimiento de la trayectoria con un offset de 4 .....                         | 113 |
| Figura 155. Implementación del punto objetivo .....  | 114 |
| Figura 156. Cálculo del ángulo objetivo .....  | 114 |
| Figura 157. Control del vehículo para el algoritmo del GPS .....                           | 114 |
| Figura 158. Función para la transmisión de los datos en el algoritmo del GPS.....          | 115 |
| Figura 159. Punto inicial de la prueba.....  | 116 |
| Figura 160. Instantánea tomada durante el test.....  | 116 |
| Figura 161. Reconducción completada .....  | 117 |
| Figura 162. Situación inicial antes del cruce.....   | 118 |
| Figura 163. Instantánea tomada durante la curva.....                                       | 118 |
| Figura 164. Imagen del vehículo después de girar.....                                      | 119 |
| Figura 165. Esquema del control completo del vehículo .....                                | 120 |
| Figura 166. Reinicio del sistema y adición de las funciones auxiliares .....               | 121 |
| Figura 167. Inicialización de las variables necesarias.....                                | 121 |
| Figura 168. Inicialización de la simulación.....   | 122 |
| Figura 169. Obtención de la ruta planeada.....   | 123 |
| Figura 170. Obtención de la imagen desde Coppelia .....                                    | 123 |
| Figura 171. Selección del algoritmo para el seguimiento de la trayectoria.....             | 124 |
| Figura 172. Bucle condicional para la detección de las señales.....                        | 124 |
| Figura 173. Implementación de la red neuronal y muestreo por pantalla .....                | 125 |

|  |     |
|--|-----|
| Figura 174. Imagen devuelta por el algoritmo de control .....                    | 125 |
| Figura 175. Transmisión de la información de Matlab a Coppelia .....             | 125 |
| Figura 176. Función auxiliar visualizaImagenCoppelia.....                        | 126 |
| Figura 177. Comprobación para saber si se ha llegado al destino.....             | 126 |
| Figura 178. Generación de la imagen resultante de la trayectoria realizada ..... | 126 |
| Figura 179. Imagen ejemplificativa tras completar el trayecto .....              | 127 |



# Índice de Tablas

---

|  |    |
|--|----|
| Tabla 1. Niveles de autonomía según el SAE .....             | 24 |
| Tabla 2. Tabla comparativa redes Deep Network Designer ..... | 40 |
| Tabla 3. Comparación redes neuronales entrenadas.....        | 70 |

# Memoria

---

## 1. Introducción, motivación y objetivos

A lo largo de este capítulo, se desarrollarán los motivos que han llevado a la realización del proyecto junto con un pequeño prólogo antes del desarrollo de este.

### 1.1. Introducción

Este Trabajo Fin de Grado (TFG) pretende desarrollar un sistema de visión inteligente para la conducción autónoma de un vehículo en entornos urbanos. Para ello, se utilizarán distintas técnicas de visión por computación tales como morfología matemática o *machine learning* junto con mecanismos de planificación de trayectoria para lograr así que un vehículo de tipo Ackerman complete el trayecto deseado de manera autónoma.

La combinación de dos de las ramas de investigación más punteras en los últimos años, tales como la conducción autónoma en los vehículos y la visión por computador, permite obtener un modelo de última generación en el cual el vehículo es capaz de adaptarse en tiempo real a cualquier cambio en el entorno. Con ello, se logra preservar la seguridad no solo de los pasajeros, sino también la de los viandantes sin intervención externa.

Mediante el análisis del entorno captado por una cámara en la parte frontal del vehículo, se ha logrado un algoritmo secuencial capaz de comprender su entorno y obtener información relevante de este para la posterior toma de decisiones. Además, para la parte de la planificación de la trayectoria y el seguimiento de esta se utilizará tanto la información del entorno como el gps del propio vehículo. La simulación, se realizará en una escena del software Coppelia Sim donde además de las distintas calles y rutas por donde circulará el vehículo, se han añadido diferentes objetos para dotar a esta escena de un mayor realismo, tales como edificios, señales, arcenes, césped...

Esta herramienta de simulación se vinculará en tiempo real con la herramienta de programación Matlab, desde donde se controlará el vehículo y se ejecutará el algoritmo diseñado para este propósito. Cabe destacar, que este algoritmo de visión se completa con la creación de diferentes scripts desde donde se podrá trabajar con cada parte por separado de manera completamente autónoma, lo que otorga a este proyecto una enorme ventaja competitiva como es la modularidad.

### 1.2. Motivación

Las dos principales motivaciones para la realización de este proyecto son, la necesidad que existe en la sociedad por incrementar la seguridad vial y la divulgación de esta rama de investigación en entornos educativos. Aunque estas puedan parecer ligeramente inconexas, están enormemente conectadas ya que no existe mejor forma de mejorar la sociedad del futuro que mediante la educación de las futuras generaciones.

### **1.2.1. Seguridad vial**

En las últimas décadas, los coches se han convertido en el principal medio de transporte y aunque cada vez son más seguros, nunca es suficiente seguridad cuando se trata de proteger vidas. Es por ello, que es enormemente conveniente la implementación y desarrollo de algoritmos que sean capaces de reemplazar al conductor o al menos reducir al máximo su implicación durante la conducción.

Aunque reducir estos accidentes no es tarea sencilla, y es necesaria la implicación de distintas herramientas de manera simultánea, cualquier avance que se pueda hacer en este sentido será más que positiva. Además, técnicas de ayuda a la conducción, similares a las desarrolladas en este proyecto, ya han sido implementadas en vehículos de nueva generación quedando demostrada su funcionalidad reduciendo los accidentes viales.

### **1.2.2. Robótica educativa**

En los últimos años, la visión inteligente y la conducción autónoma han sufrido un crecimiento exponencial y la única forma de mantenerse al día y no desactualizarse, es la divulgación de estos conceptos en todo tipo de entornos.

Siendo conscientes de que este desarrollo en robotización e inteligencia artificial continuará aumentando, es importante promover estos conocimientos desde edades tempranas para ir familiarizando a las futuras generaciones con términos tales como red neuronal, visión por computador o inteligencia artificial.

Además, la conducción autónoma y la robótica educativa combinan diferentes áreas de aprendizaje tales como la programación, visión artificial, matemáticas o la lógica. Esto convierte a este tipo de proyectos multidisciplinarios, en grandes incorporaciones para cualquier plan de estudios, ya que lo aprendido en ellos podrá ser aplicado en el futuro en trabajos o estudios de cualquier otro campo.

Como consecuencia de ello, este tipo de programas curriculares están cada vez más presentes en los centros educativos europeos, donde se instruye a los estudiantes en este campo [1]. Con este proyecto, se obtendrá un algoritmo de control mediante la combinación de disparidad de técnicas, las cuales pueden ser modificadas o ampliadas para centrar el aprendizaje o el trabajo en el aspecto o aspectos concretos que se quieran desarrollar.

Por otra parte, una de las principales ventajas de este tipo de programas es el bajo coste que estos suponen, ya que la mayoría de los programas de simulación, entre ellos los que se han utilizado en este proyecto, ofrecen versiones gratuitas para entornos educativos. De esta forma, el usuario puede empezar a familiarizarse con estos programas sin tener la gran barrera de entrada que podría suponer el coste de estos.

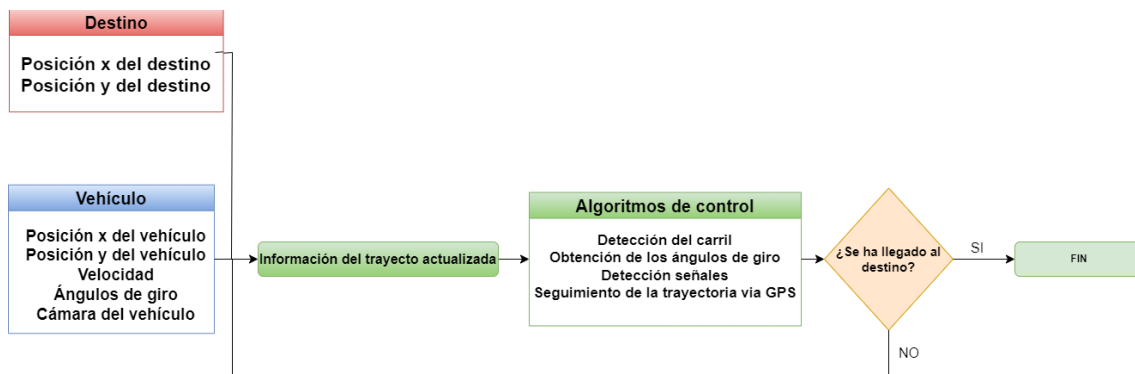
### 1.3. Objetivos

El objetivo general de este proyecto es el desarrollo de un sistema que permita la investigación y la docencia en el campo de la conducción autónoma mediante un entorno simulado que reduzca enormemente los costes. Para lograr esto, será necesario cumplir con los siguientes subobjetivos:

- Diseño e implementación de un entorno de simulación que permita recrear un entorno urbano con señales, curvas, semáforos, rotondas, edificios, carriles...
- Implementación del conexionado entre distintos programas de simulación en tiempo real
- Creación de una red neuronal para analizar de manera inteligente el entorno y actuar en concordancia con él.
- Creación de algoritmos de visión por computación para el guiado del vehículo
- Integración de un sistema de GPS para el posicionamiento y la planificación de rutas

### 1.4. Esquema general

A continuación, se adjunta un esquema general del algoritmo utilizado y diseñado en este proyecto para el correcto funcionamiento de un vehículo autónomo.



*Figura 1. Esquema general del algoritmo*

Aunque los detalles de los algoritmos de control y su funcionamiento se explicarán más en detalle en sus respectivos capítulos, en el esquema de la figura superior, se puede observar cómo el objetivo de este modelo es el control guiado del vehículo hasta llegar al destino deseado. Por ello, se comprobará en cada iteración si se ha finalizado el trayecto y en caso de ser así se detendrá la ejecución.

## 2. Estado del Arte

A lo largo de este capítulo, se hará un recorrido a lo largo de la conducción autónoma desde sus orígenes, hasta la situación actual. Con esto se logrará poner en perspectiva, y especialmente en valor, todas aquellas investigaciones realizadas a lo largo de los años tanto con fondos públicos como privados.

Cabe destacar, que gran cantidad de las investigaciones que se mencionan en este capítulo, especialmente las académicas, han sido publicadas como código abierto para que todo el mundo tenga acceso a ellos sin ningún coste. Este hecho, contribuye enormemente al desarrollo y la divulgación de estas técnicas ya que al ser públicas, cualquier persona puede trabajar en ellas y así mejorarlas de cara al futuro.

Los vehículos autónomos, han sido desde el siglo XX, una idea futurista más propia de las películas de ciencia ficción que del mundo real, sin embargo, a medida que la tecnología ha ido evolucionando, esta idea ha ido cogiendo fuerza hasta ser actualmente, uno de los campos de investigación más candentes en los últimos años. En la siguiente figura, se ilustra la sensorización estándar con la que cuentan los vehículos modernos y el funcionamiento de esta al aparcar para indicar la distancia hasta posibles obstáculos.



*Figura 2. Ilustración sobre la captación de información sobre el entorno [2]*

La primera vez que alguien soñó con la conducción autónoma, fue en 1925, cuando los habitantes de Nueva York fueron testigos de lo que hasta entonces no era más que una fantasía, un vehículo circulando por la ciudad sin ningún conductor en su interior [3]. Aunque en aquella ocasión, el coche estaba controlado por radiocontrol, esto supuso un hecho histórico desencadenando un cambio de paradigma en el campo de la automoción.

Ya en 1939, y con motivo de la Exposición Mundial de Nueva York, Norman Bel Geddes presentó su exposición Futurama, donde se expuso una maqueta sobre las posibles ciudades del futuro. En ella, se podían observar cómo automóviles eléctricos circulaban de manera autónoma y con energías renovables por la ciudad mediante campos electromagnéticos en la carretera [4] como se puede observar en la siguiente figura.



*Figura 3. Las calles del futuro según la maqueta Futurama (1939)*

Aunque estas hipótesis no fuesen más que cábalas y actualmente no representen de manera fidedigna la realidad, ejemplifican el inexorable deseo de la humanidad por hacer de la conducción autónoma una realidad. Con este objetivo, a lo largo de los años se ha ido dotando a los vehículos con más tecnología, tanto por software como por hardware, para percibir al máximo el entorno mediante sensores tales como radar, LIDAR, GPS, ultrasonidos, infrarrojos o sensores de visión.

Este conjunto de sensores y actuadores, no se implementan con este único fin, sino que contribuyen enormemente al aumento de la seguridad vial por lo que este campo de investigación cobra aún más importancia.

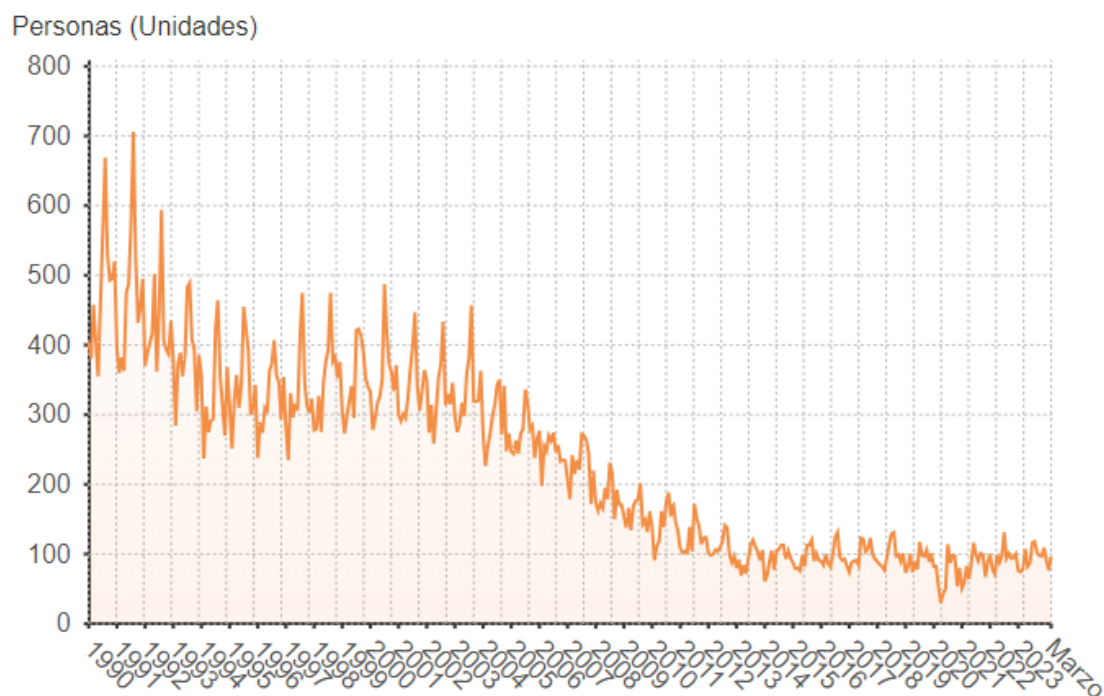
Con el paso de los años y la implementación de los Sistemas Avanzados de Asistencia a la Conducción (ADAS) [5], se han reducido progresivamente el número de accidentes y se espera que entre 2022 y 2038 se eviten 25000 muertes y 140000 heridos graves en la Unión Europea gracias a estos sistemas.

En la siguiente gráfica, se muestra la contundente reducción de los accidentes mortales en España en los últimos 34 años.



## Víctimas mortales de accidentes de tráfico desde 1990

(Datos a 24 horas después del accidente)



**Figura 4. Evolución de las víctima mortales en accidentes de tráfico en España [6]**

Y es que, no hay que olvidar que el error humano origina, según el Instituto Nacional de Estadística y Geografía (Inegi), al menos el 95% de los accidentes viales [7] y aunque estos ADAS no consten como conducción autónoma, su principio de funcionamiento es el mismo, obtener la mayor información del entorno para tomar, o aconsejar al conductor, las mejores decisiones posibles.

La principal diferencia, es que mientras los ADAS no intervienen más que como asesores o en situaciones de riesgo evidente, en la conducción autónoma el vehículo debe ser capaz de tomar todas las decisiones rutinarias, lo que sin duda, requiere una mayor capacidad de procesamiento. Independientemente del grado de protagonismo que estos sistemas tengan sobre las actividades rutinarias de conducción, es innegable que cada vez la tecnología está más presente en los vehículos, ya sea en forma de asistente o de piloto.

Para clasificar de manera precisa el grado de autonomía de los vehículos, la Sociedad de Ingenieros Automotrices (SAE), creó un estándar para clasificar los vehículos según su autonomía en 6 niveles distintos [8]. Entre los niveles 0 y 1, se requiere un grado de atención por parte del conductor muy alta, entre los niveles 2-4 el vehículo ya será un poco autónomo aunque sigue requiriendo la atención del usuario y no deja de ser un asesor, y no será hasta el nivel 5 donde se considera el vehículo completamente autónomo.

A continuación, se adjuntan dos tablas que ilustran perfectamente estas distinciones.

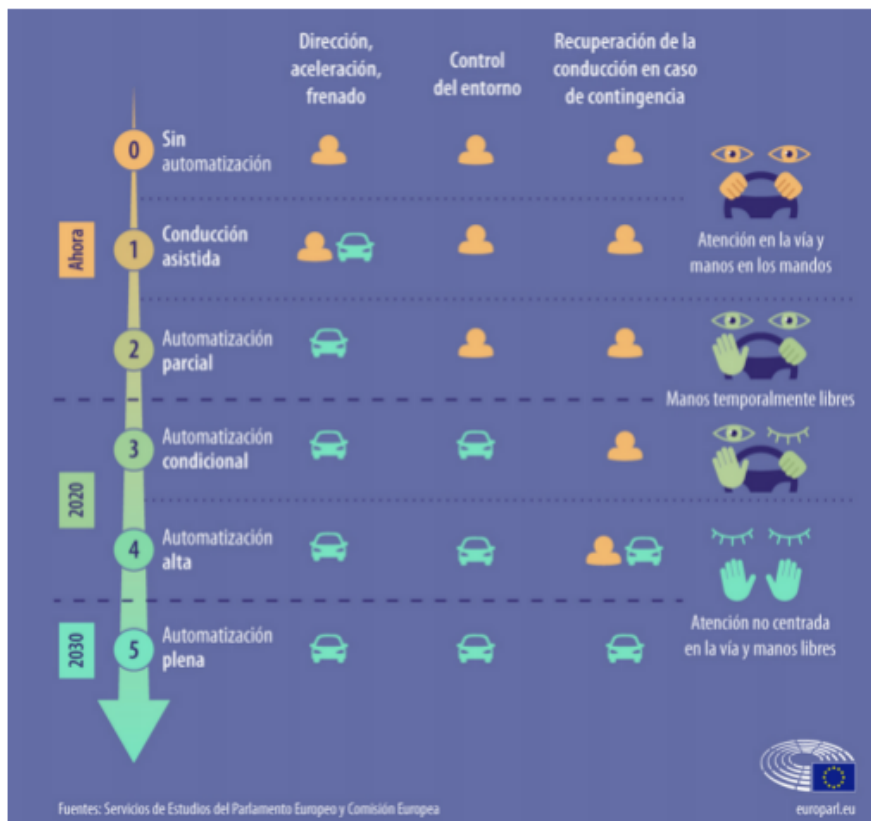


Figura 5. Niveles de conducción autónoma

| Nivel | Definición              | Resumen  |
|-------|-------------------------|--|
| 0     | Solo el conductor       | El coche no consta de un sistema automático, las tareas de conducción las realiza únicamente el conductor                                    |
| 1     | Asistente de conducción | El conductor realiza las tareas de conducción aunque el sistema puede asesorarle en determinadas situaciones.                                |
| 2     | Conducción compartida   | El vehículo puede moverse sin intervención humana aunque el conductor debe estar pendiente en todo momento.                                  |
| 3     | Autonomía controlada    | En caso de fallo el sistema es capaz de informar al usuario con la suficiente antelación como para que este pueda responder en concordancia. |
| 4     | Alto nivel de autonomía | Similar al nivel 3, aunque en este caso si el conductor no responde, el sistema es capaz de reducir el riesgo al mínimo.                     |
| 5     | Autonomía completa      | El vehículo no requiere la atención de ningún usuario y es capaz de circular libremente en cualquier tipo de entornos.                       |

Tabla 1. Niveles de autonomía según el SAE

El creador del primer vehículo autónomo de la historia fue Erns Dickmanns, quien en 1986 compró una furgoneta y la dotó con ordenadores, cámaras y todo tipo de sensores para en 1987 lograr hacerla circular de manera autónoma por una autopista vacía [9]. El rutinario sistema con el que contaba aquella caravana durante las primeras pruebas se adjunta en la siguiente figura.



*Figura 6. Interior de la furgoneta de Dickmanns*

Posteriormente, y tras la inversión por parte del fabricante de automóviles alemán Daimler, se logró obtener unos fondos de 749 millones para el proyecto conocido como Eureka PROMETHEUS Project (*PROgramme for a European Traffic of Highest Efficiency and Unprecedented Safety*), la mayor inversión jamás vista en una investigación de este tipo [10]. Todos estos esfuerzos, dieron su resultado cuando en 1994 se logró hacer circular a un vehículo a 130 km/h mientras cambiaba de carril para esquivar coches mediante un ordenador de a bordo, basándose simplemente en visión, que controlase el volante y los pedales del acelerador y el freno.

En la actualidad, estos sistemas son mucho más complejos y los líderes en este sector son los principales fabricantes de automóviles y compañías tecnológicas. Cabe destacar entre todas ellas, el proyecto Waymo, perteneciente al grupo Google y que ya está disponible para utilizar como taxi para los habitantes de Los Ángeles y Austin, logrando excepcionales resultados [11].

Como se puede observar a simple vista comparando la figura 6 con la siguiente figura, la tecnología empleada ha aumentado notablemente pero el principio de funcionamiento y el objetivo siguen intactos.



*Figura 7. Vehículo Waymo en funcionamiento [12]*

En la actualidad, todas estas técnicas y algoritmos son combinados entre si para usar lo mejor de cada una y ofrecer así la mejor experiencia posible al usuario tanto en términos de seguridad como de confort.

## 2.1. Investigaciones públicas previas

Cabe mencionar, el importante trabajo desarrollado sobre la conducción autónoma por numerosos equipos independientes y universidades. Gracias a ellos, este campo de investigación ha crecido exponencialmente en los últimos años hasta el punto de tener la primera competición de monoplazas autónomos de la historia, la “Abu Dhabi Autonomous Racing League” donde los vehículos autónomos pueden alcanzar los 300 km/h [13].

Algunos trabajos que merece la pena destacar, dentro de la infinidad de grandes investigaciones que se han desarrollado en estos entornos son los siguientes:

- **Conducción autónoma:** Un gran trabajo en este ámbito, es el realizado en la Universidad de Coruña, donde se utilizó un pequeño robot para desarrollar algoritmos de control que le permitiesen moverse de manera autónoma [14].
- **Detección de carril y señales:** Trabajo orientado a la competición Seat Autonomous Driving y donde se desarrolla el reconocimiento de señales y carriles [15]
- **Detección de carril:** Tesis doctoral sobre la detección de carril en carretera [16].
- **Coppelia:** Trabajo realizado en Coppelia con una extensa investigación sobre las oportunidades que este software ofrece [17].
- **Redes neuronales:** Trabajo Fin de Máster sobre la implementación de redes neuronales para vehículos autónomos [18].

Aunque todos estos trabajos han sido fundamentales para el desarrollo de este proyecto, es de especial importancia recalcar los dos trabajos sobre los que está basado este trabajo. El primero de ellos, es el que se utilizará como base para la parte de la simulación en Coppelia ya que utilizando Python, se ha logrado crear una escena de CoppeliaSim con un vehículo funcional [19]. Esta escena, se adaptará para simular el comportamiento del modelo en una ciudad mediante la adición o cambio de algunas circunstancias como por ejemplo la inclusión de curvas para probar la detección de carril en esos tramos concretos. De esta forma se buscará probar el modelo en unas circunstancias lo más reales posibles y así asegurar la escalabilidad de este.

El otro trabajo imprescindible para la realización de este modelo es un trabajo de fin de grado de la Universidad Politécnica de Valencia, donde se planteó por primera vez la simulación de un vehículo autónomo en Coppelia mediante su control desde Matlab [20].

Con este trabajo, se desea poner en común lo mejor de los trabajos mencionados anteriormente, e ir un paso más allá en la obtención de la completa autonomía en los vehículos. Para ello, se avanzarán o mejorarán las técnicas mencionadas en dichas investigaciones y se añadirán técnicas nuevas que otorguen a este sistema ventajas antes situaciones extremas o una mayor robustez.

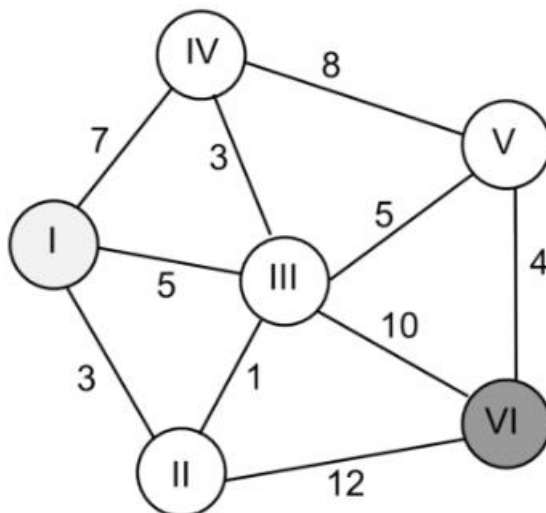
## 2.2. Algoritmos de planificación de ruta

Por su parte, la planificación de rutas es un campo mucho más maduro, y es que desde el desarrollo del GPS en 1960 por parte del Departamento de Defensa de Estados Unidos [21], este campo de investigación se ha desarrollado enormemente. A pesar del temprano descubrimiento de esta tecnología, no fue hasta 1991 cuando esta fue liberada para su uso civil [22].

En 1996, MapQuest lanzó el primer planificador de rutas y debido a su temprano éxito, grandes empresas como Google o Yahoo! desarrollaron sus propios modelos. Estos primeros modelos utilizaban mapas digitales y eran relativamente imprecisos pero en los últimos años, gracias a la información en tiempo real, aplicaciones como Google Maps [23] se han hecho enormemente populares y útiles para la población.

Los algoritmos de planificación de ruta son algoritmos basados en el desarrollo de técnicas que permiten a los vehículos no solo generar un trayecto eficaz para llegar a un destino, sino optimizarlo para que este sea lo más eficiente posible, ya sea en términos de contaminación, tráfico, tiempo... Estos, están basados en grafos, que es un esquema de nodos, puntos (origen / destino) y rutas (conexión entre nodos), según estos grafos, se genera un árbol de caminos que conecten el nodo origen con el resto de los nodos y elija la ruta más corta según el criterio que se busque en dicha implementación.

Uno de los primeros algoritmos de planificación de ruta desarrollados es el Algoritmo de Dijkstra [24], aunque en la actualidad hay multitud de métodos con el fin de adaptar estos lo máximo posible a cada problema particular. A continuación, se adjunta una imagen ejemplificativa de un grafo utilizado en este ámbito.



*Figura 8. Grafo ejemplificativo [25]*

Este tipo de algoritmos, están presentes no solo en automoción sino que aparecen en numerosos y diversos ámbitos profesionales debido a su gran eficiencia a la hora de comparar costes con procesos. Sin embargo, en entornos muy cambiantes como los de la conducción autónoma, la planificación de los movimientos no solo debe ser precisa sino también rápida, por lo que son necesario algoritmos enormemente adaptativos.

Por ello, los algoritmos utilizados en conducción autónoma se orientan no solo a detectar los cambios en el entorno de manera rápida, sino a aprender de ellos, lo que permite anticipar situaciones y salvar circunstancias de riesgo con la antelación suficiente. Estos algoritmos, han mejorado exponencialmente mediante la inclusión del *Machine Learning* y la inteligencia artificial en ellos [26], el conjunto de algoritmos utilizados para este tipo de trabajo es:

- **Matriz de decisión:** Matriz utilizada para tomar decisiones basándose en ciertas condiciones, útil para barajar diferentes alternativas en situaciones de alta incertidumbre
- **Matriz de agrupamiento:** Matriz para la agrupación de diferentes objetos mediante similitud en un grupo (un clúster), algunos algoritmos típicos de esta matriz de agrupamiento son K-means o DBSCAN.
- **Matriz de reconocimiento de patrones:** Es la principal matriz utilizada en el machine learning y en las redes neuronales para entrenar modelos según la detección de patrones comunes. Estos patrones son especialmente útiles de cara a poder anticiparse a futuras situaciones.
- **Matrices de regresión:** Se utilizan algoritmos de regresión para predecir una variable de salida basándose en el conjunto de parámetros de entrada que se tengan, cuanto mayor sea el número de inputs menor será la incertidumbre y por lo tanto mayor será la precisión del sistema.<sup>7</sup>

En este aspecto, cabe destacar el trabajo de la Universidad Técnica de Múnich en la creación de un algoritmo que sea eficiente en entornos urbanos reales [27], aunque hay gran variedad de ellos y la implementación de uno u otro depende principalmente de los requerimientos de cada proyecto en particular.

### 2.3. Algoritmos de visión por computador

La visión por computador, se entiende como el proceso de adquirir, procesar y analizar información visual obtenida mediante cámaras o sensores de visión, con el objetivo de comprender el entorno en su totalidad para poder actuar en consecuencia y evitar posibles riesgos. En los últimos años, la visión por computación se ha visto completamente transformada por el empuje imparable de la inteligencia artificial, y más en particular, por las redes neuronales convolucionales (CNN).

Estas redes, permiten al sistema aprender de una base de datos de tal forma que cuando se enfrente a un problema similar en el futuro esta sea rápidamente capaz de identificar el entorno de manera confiable y segura.

Estudios realizados en los últimos años [28], demuestran como esta capacidad para aprender de casos anteriores sin requerir ser reprogramados, supone una mejora sustancial, especialmente en escenarios futuribles, respecto a modelos anteriores. Es por ello, que cada vez son más los fabricantes que integran este tipo de técnicas en sus vehículos, como es el caso de Tesla [29], donde el *Deep learning* está tomando cada vez más protagonismo dentro de sus coches.

Sin embargo, y a pesar de las infinitas oportunidades que el *machine learning* ofrece, esto no significa que el resto de los procedimientos sean ineficientes, y es que, en tareas repetitivas o con poca incertidumbre, los métodos tradicionales de visión por computación siguen dando excelentes resultados. Es por ello, que el mejor camino de cara a lograr la conducción autónoma completa es la combinación de todos estos métodos y algoritmos para obtener lo mejor de cada uno sin cernirse exclusivamente a uno de ellos.

Cabe mencionar, que al ser este un campo tan novedoso y complejo, ningún sistema por sí mismo puede considerarse inequívocamente como mejor ya que sus resultados dependerán enormemente de las necesidades concretas que se tengan. Teniendo en cuenta esto, el conjunto de investigaciones académicas y profesionales proporcionan una fuente fundamental de información para la consecución de lo que en aquella Exposición Universal de Nueva York no era más que un delirio.

### 3. Especificaciones de diseño y conocimientos previos

A lo largo de este capítulo se expondrán de forma detallada, todas las herramientas utilizadas para el diseño e implementación del modelo realizado. Además, se explicarán los conocimientos previos necesarios para adentrarse dentro de este y poder utilizarlo de manera plena.

Se requerirá un conocimiento mínimo de programación en Lua y especialmente de programación en Matlab, para utilizar el algoritmo de control encargado de la conducción autónoma. Además, será necesario algún conocimiento básico sobre el modelo pinhole que se utilizará para el cálculo de las distancias y de algunas de las aplicaciones que Matlab ofrece, tales como Deep Network Designer o Color Thresholder aunque estas podrán ser modificadas por técnicas análogas. A su vez, será positivo tener un conocimiento básico sobre la planificación de trayectoria y el funcionamiento de este tipo de modelos.

Cabe destacar, que de cara a un completo entendimiento de los algoritmos de control se requieren conocimientos sólidos y extensos de *Machine Learning*, morfología matemática y segmentación de imágenes. Junto con lo mencionado anteriormente, serán de gran ayuda conocimientos básicos de física y matemáticas para el entendimiento de los algoritmos utilizados para el seguimiento de la trayectoria y el control de un vehículo tipo Ackerman.

#### 3.1. Coppelia Sim

Coppelia Sim, anteriormente conocido como Virtual Robot Experimentation Platform (V-REP), es un software de simulación en 3D propiedad de Coppelia Robotics [30]. Este software, permite crear escenarios personalizados donde poder probar los modelos que queramos en situaciones que de ser testeadas en el mundo real requerirían un elevadísimo coste.

Una de las mayores ventajas que este programa ofrece, es la enorme versatilidad disponible, ya que se basa en una arquitectura de control distribuido [31], es decir, cada objeto o modelo puede ser controlado de manera individual mediante script embebidos, por complementos como plugins o add-ons, por un nodo ROS o un nodo BlueZero o una API remota.

Estas opciones difieren entre sí en las siguientes características.

- **Scripts embebidos:** Es el método más directo ya que la programación se realiza mediante scripts hijos dentro de CoppeliaSim y programados en LUA. Son de especial utilidad de cara a personalizar la apariencia o funcionalidad de la escena.
- **Plugins:** se utiliza en combinación con el primer método o bien para otorgar a CoppeliaSim una funcionalidad especial tal como la conexión con sistemas físicos.
- **Complementos (add-ons):** Sistema similar a los scripts embebidos aunque estos están centrados en objetos o modelos particulares, permiten el uso de funciones y están escritos en su totalidad en LUA.
- **Nodo ROS:** es el método más eficiente para conectar Coppelia con una aplicación externa.
- **Nodo BlueZero:** similar al nodo ROS, aunque es menos común que el nodo ROS y por lo tanto dispone de menos documentación a pesar de ser enormemente multidisciplinar.
- **API remota:** utiliza comandos API remotos para conectar Coppelia con programas externos proporcionando una configuración estándar para la comunicación independientemente del lenguaje de programación utilizado.

A continuación, se adjunta una tabla comparativa de las 6 alternativas mencionadas anteriormente [32].

|  | Embedded script              | Add-on / sandbox script      | Plugin   | Client application                               | Remote API client                               | ROS / ROS2 node                 | ZeroMQ node                     |
|--|------------------------------|------------------------------|--|--|---|---------------------------------|---------------------------------|
| Control entity is external (i.e. can be located on a robot, different machine, etc.) | No                           | No                           | No   | No   | Yes   | Yes                             | Yes                             |
| Supported programming language   | Lua, Python                  | Lua, Python                  | C/C++  | C/C++, Python                                    | C/C++, Python, Java, JavaScript, Matlab, Octave | Any <sup>1</sup>                | Any                             |
| Code execution speed   | Relatively fast <sup>2</sup> | Relatively fast <sup>2</sup> | Fast   | Fast   | Depends on programming language                 | Depends on programming language | Depends on programming language |
| Communication lag  | None <sup>3</sup>            | None <sup>3</sup>            | None   | None   | Yes   | Yes                             | Yes                             |
| Communication channel  | Python: ZeroMQ <sup>3</sup>  | Python: ZeroMQ <sup>3</sup>  | None   | None   | ZeroMQ or WebSockets                            | ROS / ROS2                      | ZeroMQ                          |
| Control entity can be fully contained in a scene or model, and is highly portable    | Yes                          | No                           | No   | No   | No  | No                              | No                              |
| Stepped operation <sup>4</sup>   | Yes, inherent                | Yes, inherent                | Yes, inherent                                    | Yes, inherent                                    | Yes   | Yes                             | Yes                             |
| Non-stepped operation <sup>4</sup>   | Yes, via threads             | Yes, via threads             | No (threads available, but API access forbidden) | No (threads available, but API access forbidden) | Yes   | Yes                             | Yes                             |

<sup>1</sup> Depends on ROS / ROS2 bindings

<sup>2</sup> Depends on the programming language, but the execution of API functions is very fast

<sup>3</sup> Lua scripts are executed in CoppeliaSim's main thread, Python scripts are executed in separate processes

<sup>4</sup> Stepped as in *synchronized* with each simulation step

**Figura 9. Tabla comparativa de los 6 métodos de control**

Estas oportunidades de interconexión entre distintos programas que Coppelia ofrece, serán fundamentales para utilizar algoritmos mucho más complejos de lo que están disponibles en Coppelia y controlar así la escena en tiempo real. Para este proyecto, se combinarán los script embebidos para el control de los objetos del escenario tales como los semáforos o el



funcionamiento básico del vehículo mientras que se utilizarán el nodo BlueZero para realizar la conexión Matlab-Coppelia y una API externa para obtener la información de Coppelia que se procesará posteriormente en Matlab.

Además, este software es compatible con Windows, Linux y macOS y tiene 3 versiones distintas:

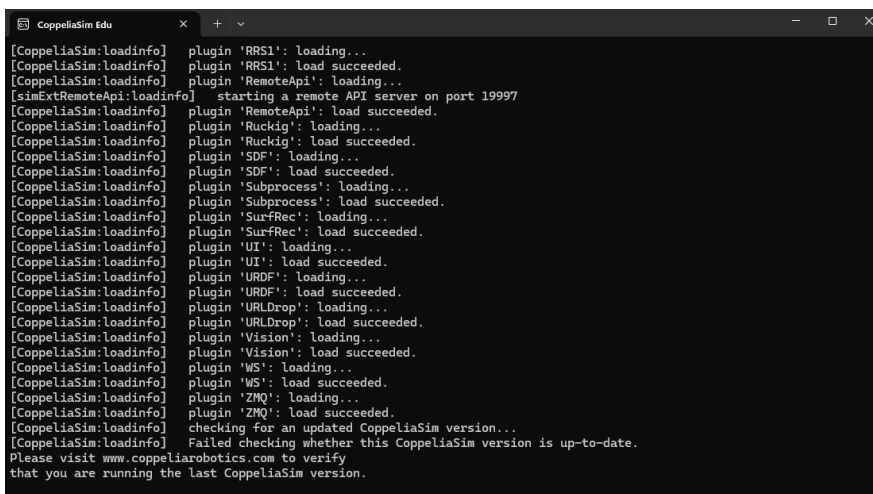
- **CoppeliaSim Player:** versión estándar, gratuita para todos los públicos.
- **CoppeliaSim Pro:** versión comercial, cuenta con todas las opciones aunque es de pago.
- **CoppeliaSim Edu:** misma versión que la Pro solo que disponible de forma gratuita para usos educativos.

Coppelia Robotics, ofrece en su web un extenso manual [32] sobre este software lo que facilita enormemente el aprendizaje de este programa y rompe con las barreras iniciales que este podría suponer en entornos educativos. A continuación, se describirán alguno de los aspectos más importantes de este programa para ofrecer una idea general del funcionamiento de este, y lograr así, una mayor comprensión del entorno de simulación antes de entrar en la parte técnica del proyecto.

### 3.1.1. Interfaz de usuario

Coppelia Sim, está compuesta por dos ventanas que se ejecutan de forma simultánea y se explican a continuación:

- **Ventana de consola:** muestra los complementos cargados y si el proceso de encendido ha sido completado con éxito, aunque no tiene mayor uso durante la ejecución del programa exceptuando el uso del comando print (en Lua) para mostrar la información deseada. Se pueden crear varias ventanas simultáneas para mostrar en cada una de ellas información en tiempo real, la siguiente figura muestra un ejemplo de ventana de consola ejemplificativa.



```

[CoppeliaSim:loadinfo] plugin 'RRS1': loading...
[CoppeliaSim:loadinfo] plugin 'RRS1': load succeeded.
[CoppeliaSim:loadinfo] plugin 'RemoteApi': loading...
[SimExtRemoteApi:loadinfo] starting a remote API server on port 19997
[CoppeliaSim:loadinfo] plugin 'RemoteApi': load succeeded.
[CoppeliaSim:loadinfo] plugin 'Ruckig': loading...
[CoppeliaSim:loadinfo] plugin 'Ruckig': load succeeded.
[CoppeliaSim:loadinfo] plugin 'SDF': loading...
[CoppeliaSim:loadinfo] plugin 'SDF': load succeeded.
[CoppeliaSim:loadinfo] plugin 'Subprocess': loading...
[CoppeliaSim:loadinfo] plugin 'Subprocess': load succeeded.
[CoppeliaSim:loadinfo] plugin 'SurfRec': loading...
[CoppeliaSim:loadinfo] plugin 'SurfRec': load succeeded.
[CoppeliaSim:loadinfo] plugin 'UI': loading...
[CoppeliaSim:loadinfo] plugin 'UI': load succeeded.
[CoppeliaSim:loadinfo] plugin 'URDF': loading...
[CoppeliaSim:loadinfo] plugin 'URDF': load succeeded.
[CoppeliaSim:loadinfo] plugin 'URLDrop': loading...
[CoppeliaSim:loadinfo] plugin 'URLDrop': load succeeded.
[CoppeliaSim:loadinfo] plugin 'Vision': loading...
[CoppeliaSim:loadinfo] plugin 'Vision': load succeeded.
[CoppeliaSim:loadinfo] plugin 'WS': loading...
[CoppeliaSim:loadinfo] plugin 'WS': load succeeded.
[CoppeliaSim:loadinfo] plugin 'ZMQ': loading...
[CoppeliaSim:loadinfo] plugin 'ZMQ': load succeeded.
[CoppeliaSim:loadinfo] checking for an updated CoppeliaSim version...
[CoppeliaSim:loadinfo] Failed checking whether this CoppeliaSim version is up-to-date.
Please visit www.coppeliarobotics.com to verify
that you are running the last CoppeliaSim version.
  
```

*Figura 10. Ventana de consola de CoppeliaSim*

- **Ventana de aplicación:** es la ventana principal, en ella está la escena donde se realiza la simulación junto con todas las herramientas y objetos que se van a utilizar, a continuación, se muestra la ventana de aplicación de la escena de este modelo.

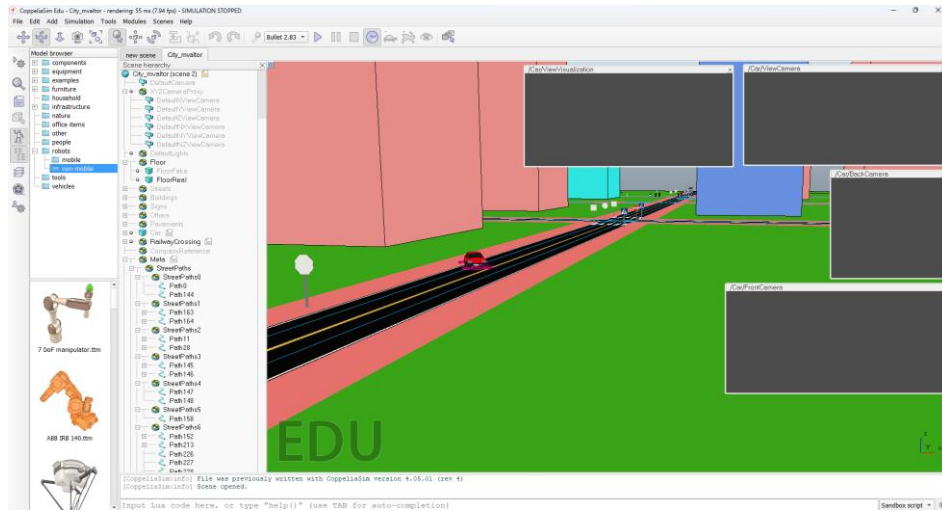


Figura 11. Ventana de aplicación de CoppeliaSim

Dentro de la ventana de aplicación, hay distintas herramientas que facilitan el manejo del programa y de la simulación, a continuación, se incluye una imagen de la ventana de aplicación con cada funcionalidad discernida para su posterior explicación.

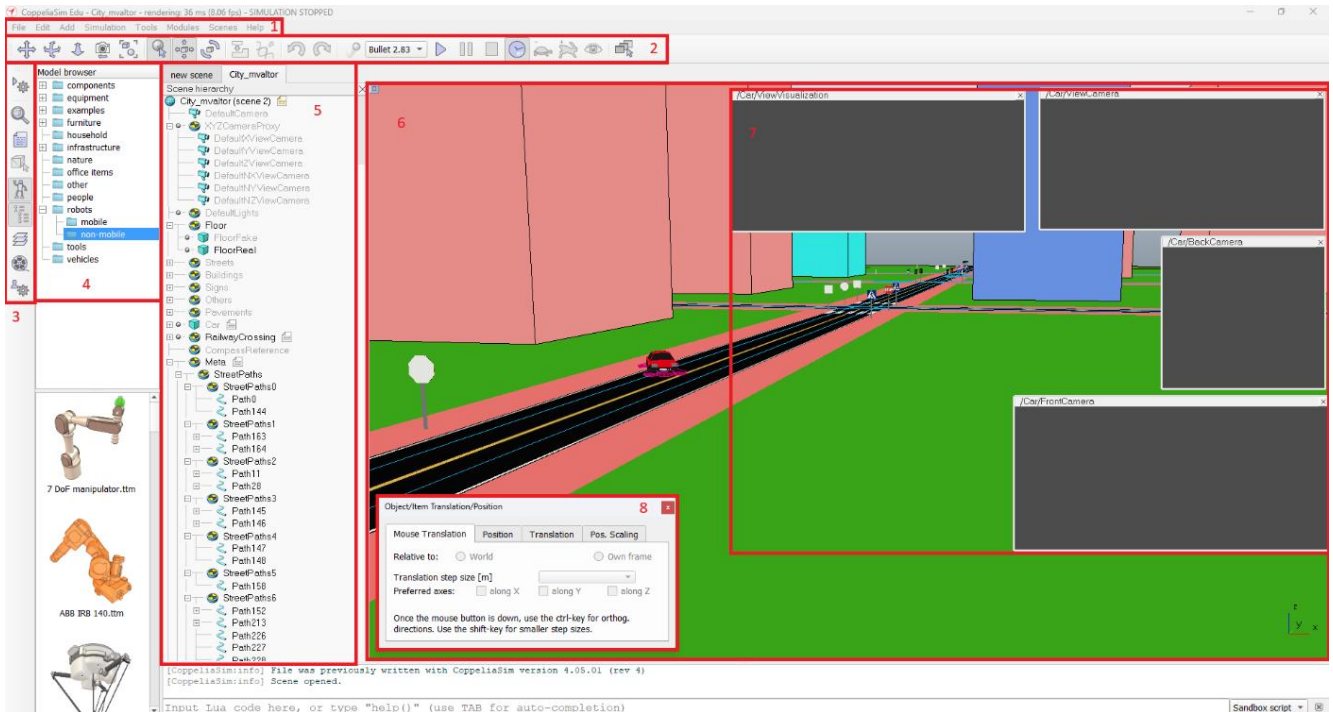


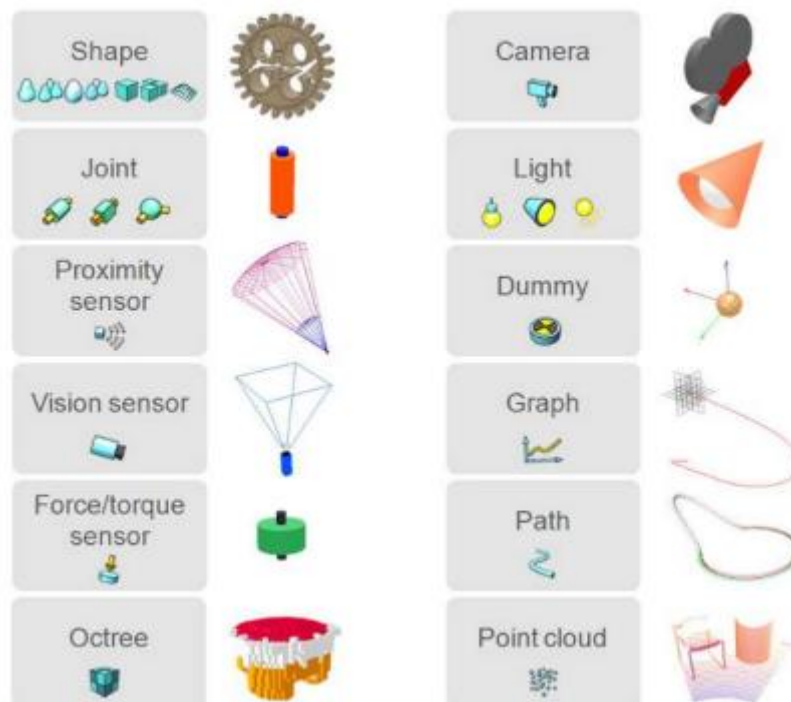
Figura 12. Ventana de aplicación dividida en partes

Aunque la mayoría de las ventanas se abren de manera flotante, es importante conocer donde se puede encontrar cada herramienta para poder optimizar este software.

1. **Barra de menú general:** Agrupa todas las funciones del software en grupos generales.
2. **Barra de herramientas:** Conjunto de herramientas que permite interactuar con la escena en tiempo real mediante el movimiento de la cámara principal, los objetos o la configuración de la ejecución.
3. **Barra de herramienta flotante:** Funcionalidad similar a la anterior pero permite el libre movimiento de esta o su eliminación, está más enfocada a la personalización de la interfaz gráfica del programa.
4. **Model Brower:** Conjunto de objetos y modelos puestos a disposición por CoppeliaSim para realizar simulaciones, incluye desde robots o personas a objetos de mero adorno.
5. **Scene hierarchy:** Muestra la jerarquía de los objetos de la simulación, permite observar de manera rápida y visual la forma en la que están agrupados los objetos y la relación entre estos.
6. **Escenario:** Lugar del software de simulación donde se puede observar el escenario utilizado para la simulación.
7. **Ventanas flotantes:** Ventanas añadidas para observar en tiempo real las imágenes devueltas por las cámaras o los sensores de visión añadidos a la escena.
8. **Cuadros de diálogo:** Se generan mediante la barra de herramientas y son las opciones mediante las cuales se interactúa con la escena de manera precisa.

### 3.1.2. Modelos y objetos

CoppeliaSim ofrece una gran variedad de componentes y modelos para crear las simulaciones necesarias y recrear de manera realista las condiciones dinámicas del mundo real, a continuación, se adjunta una figura con todas las categorías de objetos que se pueden añadir a la escena.



*Figura 13. Categoría de objetos de CoppeliaSim*

Todos estos objetos, pueden ser modificados por el usuario para recrear las características que sean necesarias, aunque todos ellos son de gran utilidad, en este caso se explicará en detalle solo aquellos que se utilicen en el trabajo.

- **Shape:** Conjunto de formas básicas que se pueden combinar y personalizar para formar objetos más complejos.
- **Camera:** Simula la instalación de una cámara, se utilizará para realizar el algoritmo de control del sistema.
- **Dummy:** Permiten seguir trayectorias o caminos a los objetos de la simulación.
- **Path:** Conjunto de dummies que forman cada camino, pueden ser rectos o curvos según la disposición de los dummies que lo compongan.

Mediante la combinación de estos objetos, se podrá simular prácticamente cualquier circunstancia por muy compleja que sea, por lo que poseer un dominio completo de estos y conocer las oportunidades que se ofrecen es fundamental.

### 3.1.3. Escena CoppeliaSim

La escena de CoppeliaSim utilizada en este proyecto recrea un entorno urbano en el que habrá distintas calles, una curva, varios cruces y varias rotondas de tal forma que el vehículo se podrá probar con todos los problemas con los que se enfrentaría en el mundo real.

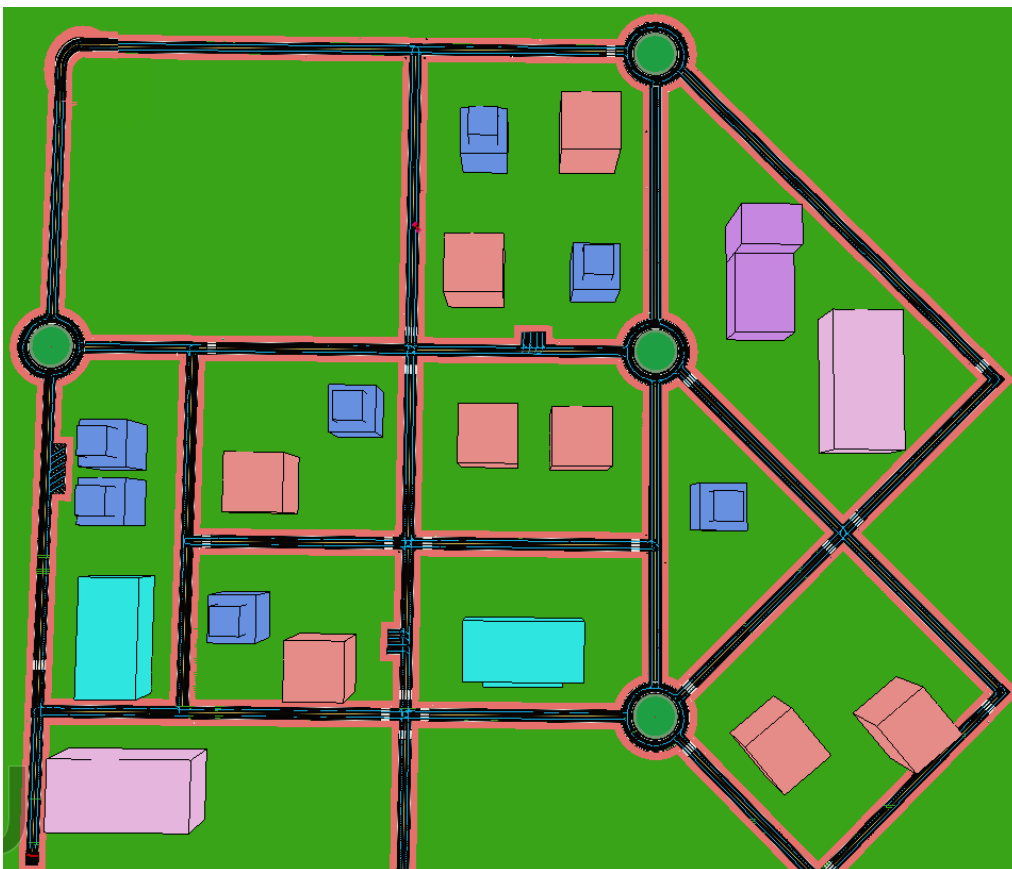


Figura 14. Escena CoppeliaSim

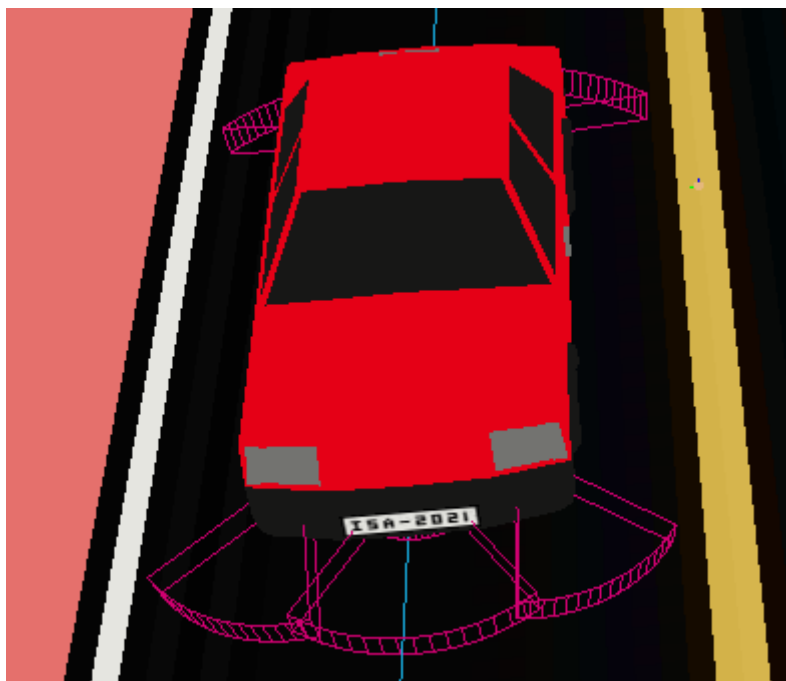
Además, también es importante mencionar la numerosa señalización que el vehículo deberá detectar, y posteriormente actuar en consecuencia, a lo largo de la simulación.

En la siguiente figura, se muestra toda aquella señalización empleada en la escena sobre la que tiene lugar la simulación.



*Figura 15. Señales escena Coppelia*

Finalmente, es importante tener en cuenta el vehículo de tipo Ackerman, encargado de realizar el trayecto deseado de manera autónoma, junto con las características de la cámara frontal de este. Gracias a esta cámara, se obtiene la imagen que posteriormente servirá al vehículo para circular, la siguiente figura, muestra el vehículo utilizado para la simulación de este proyecto y todas aquellas pruebas necesarias para su realización.








*Figura 16. Modelo del vehículo autónomo*

### 3.2. Conexión Matlab-Coppelia

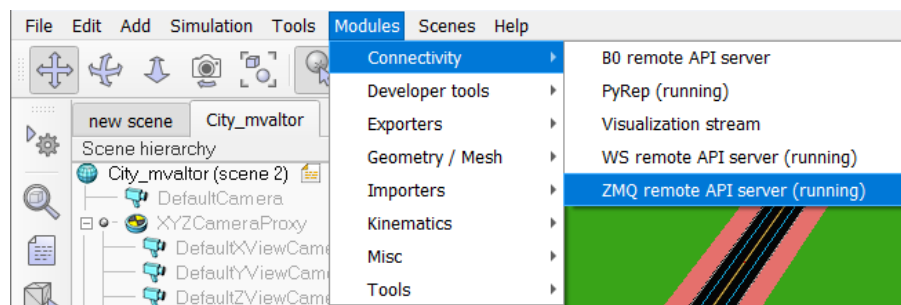
La conexión entre Coppelia y Matlab se realiza mediante Remote API [33], la cual es una librería de código abierto que permite conectarse a través de la API remota CoppeliaSim ZeroMQ siguiendo las instrucciones que se detallan a continuación:

- 1. Librerías y scripts necesarios:** Serán necesarios diferentes scripts y librerías para poder conectarse en tiempo real con la simulación e intercambiar información con esta en tiempo real. Además, será necesaria la instalación y activación de esta biblioteca en Coppelia . A continuación, se adjunta una captura de los archivos del directorio local necesarios para realizar esta conexión.

|   |                  |                       |        |
|---|------------------|-----------------------|--------|
|  cbor.m            | 28/03/2022 17:45 | MATLAB Code           | 13 KB  |
|  RemoteAPIClient.m | 28/03/2022 17:45 | MATLAB Code           | 5 KB   |
|  RemoteAPIObject.m | 28/03/2022 17:45 | MATLAB Code           | 2 KB   |
|  remoteApi.dll     | 08/02/2023 13:42 | Extensión de la ap... | 69 KB  |
|  jeromq-0.5.2.jar  | 22/05/2023 12:27 | Executable Jar File   | 459 KB |

*Figura 17. Archivos necesarios para la interconexión*

- 2. Preparación del escenario de CoppeliaSim:** Además de cargar la escena anteriormente mencionada, será necesario activar la biblioteca *SMQ remote API server* desde la ventana que se muestra en la siguiente figura.



*Figura 18. Configuración conexasión en Coppelia*

- 3. Comandos para interactuar con la escena:** Una vez realizada la configuración mencionada en los puntos anteriores ya será posible interactuar con la escena por código directamente desde la aplicación de Matlab utilizando las funciones que se detallarán en los siguientes capítulos.

### 3.3. Matlab

Este potente software de programación es de pago, aunque al igual que Coppelia Sim, ofrece una versión gratuita para los estudiantes universitarios, con completo acceso a todas las herramientas que este software ofrece. La siguiente figura muestra la oferta que Matlab hace para los estudiantes.

### Acceso en todo el campus

Es posible que su centro educativo ya ofrezca acceso a MATLAB, Simulink y otros productos complementarios mediante una infraestructura Campus-Wide License.

[Obtenga MATLAB](#)

### MATLAB and Simulink Student Suite

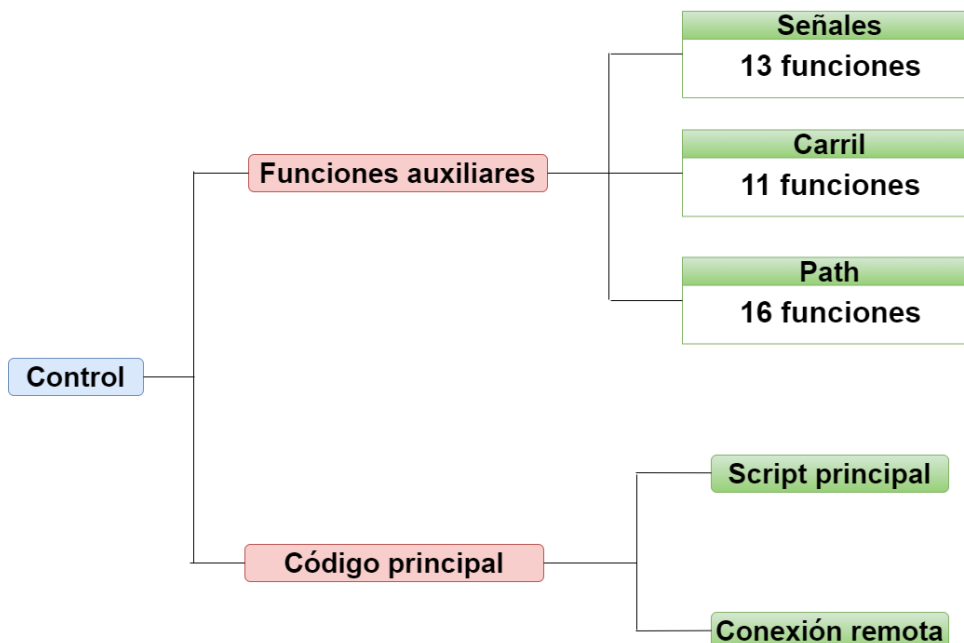
Incluye MATLAB, Simulink y 10 de los productos complementarios más utilizados, así como soporte integrado para prototipado, pruebas y ejecución de modelos en plataformas de hardware de bajo coste.

[Compre ahora](#)

[Ver todos los productos](#)

*Figura 19. Ofertas Matlab para estudiantes [34]*

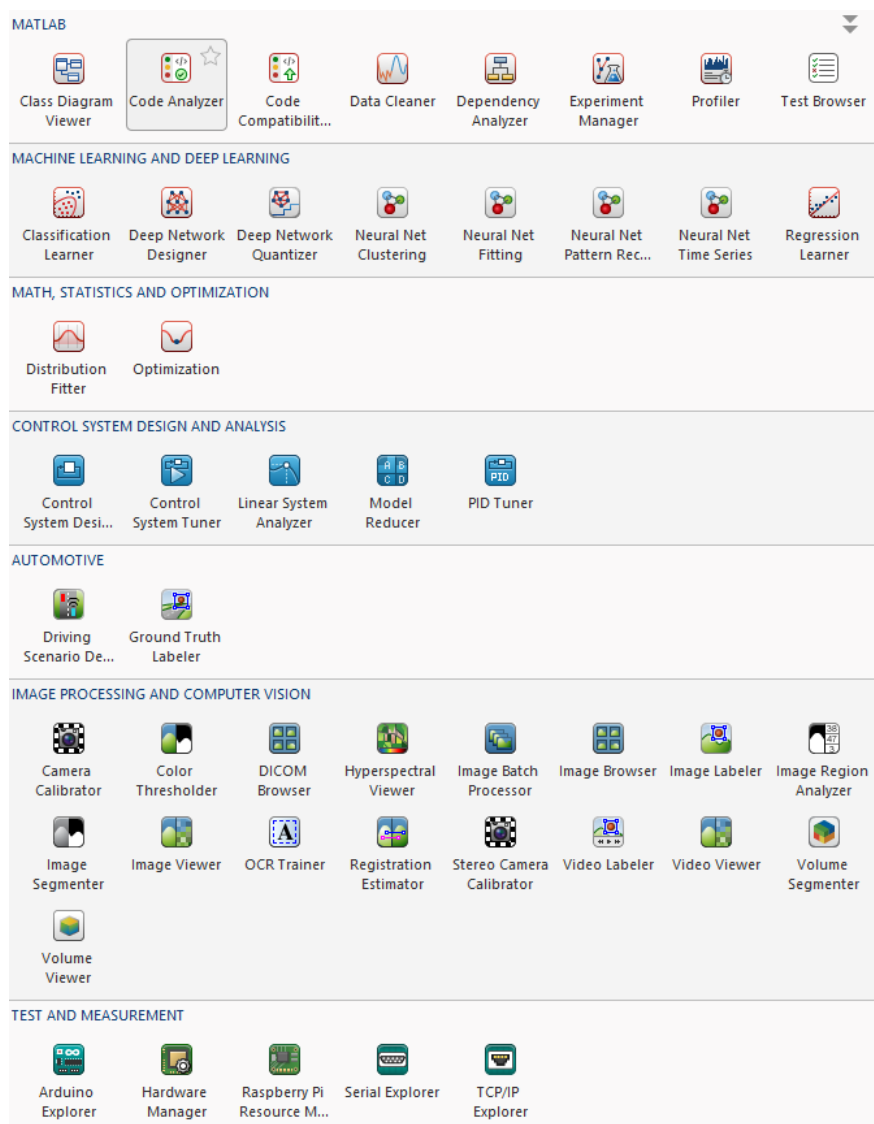
Desde este software, se realiza el algoritmo de control del modelo y se controla la simulación, para ello, se usa un script general desde donde se controla completamente la simulación en tiempo real. De cara a mantener un orden dentro del script general y hacer que este sea sencillo de seguir o modificar, se han creado funciones auxiliares para cada una de las partes del control y almacenado estas en carpetas según la siguiente disposición.



*Figura 20. Esquema arquitectura del directorio del código del proyecto*

Como se puede ver, el código para el control del sistema se ha dividido en dos grandes grupos, uno de ellos para las funciones auxiliares y el otro para el script principal y los scripts necesarios para la conexión con Coppelia. De esta forma, se consigue un código muy nítido y fácilmente moldeable lo que es esencial de cara a su difusión en entornos educativos y a su escalabilidad en proyectos más complejos.

Otra de las grandes ventajas que ofrece Matlab, es la enorme oferta de aplicaciones incluidas dentro de este programa y que convierten a este software de programación en una herramienta muy versátil y potente. Todas estas aplicaciones están incluidas dentro del paquete de Matlab mencionado con anterioridad y no supondrán ningún sobrecoste de cara al presupuesto final, a continuación, se muestra el conjunto de aplicaciones que Matlab ofrece.



*Figura 21. Conjunto de aplicaciones disponibles en Matlab*

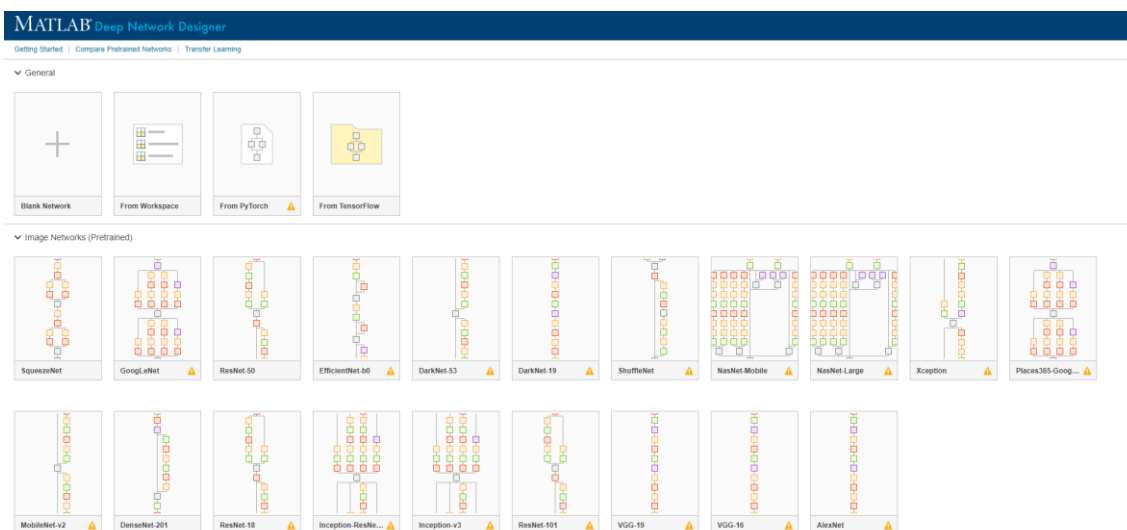
Aunque todas estas son muy útiles para distintas características, en los siguientes apartados se explican detalladamente las utilizadas en alguna de las partes del algoritmo de control.

### 3.3.1. Deep Network Designer

Esta aplicación, forma parte de la biblioteca Deep Learning Toolbox [35], Deep Network Designer, facilita enormemente la creación de una red neuronal convolucional [36] lo que se utilizará en este proyecto de cara a la detección de la señalización, una parte fundamental de cualquier vehículo autónomo.



Una de las principales ventajas de esta herramienta, es la posibilidad que ofrece de seleccionar un modelo de red concreto para utilizarlo como punto de partida, lo que simplifica y mejora los resultados en comparación con una red creada desde cero. Además, estas redes ya han sido entrenadas anteriormente con más de mil categorías de objetos y más de un millón de imágenes lo que mejorará los resultados del modelo [37]. Las redes que están disponibles dentro de esta herramienta se muestran en la siguiente figura.



**Figura 22. Redes preentrenadas disponibles en Deep Network Designer**

Es importante saber que cada red requiere un tamaño concreto de las imágenes de entrada por lo que, antes de utilizar cualquiera de ellas es importante reajustar las imágenes para que cumplan con estos parámetros y así optimizar al máximo la red neuronal que se entrene. A continuación, se adjunta una tabla comparativa de todas las redes disponibles junto con una gráfica comparativa entre la precisión y la carga de CPU de todas ellas [36].

| Nombre              | Profundidad | Tamaño (MB) | Parámetros (millones) | Tamaño de entrada de las imágenes |
|---------------------|-------------|-------------|-----------------------|-----------------------------------|
| SqueezeNet          | 18          | 5.2         | 1.24                  | 227 por 227                       |
| GoogleNet           | 22          | 27          | 7                     | 224 por 224                       |
| Inception-v3        | 48          | 89          | 23.9                  | 299 por 299                       |
| DenseNet-201        | 201         | 77          | 20                    | 224 por 224                       |
| MobileNet-v2        | 53          | 13          | 3.5                   | 224 por 224                       |
| ResNet-18           | 18          | 44          | 11.7                  | 224 por 224                       |
| ResNet-50           | 50          | 96          | 25.6                  | 224 por 224                       |
| ResNet-101          | 101         | 167         | 44.6                  | 224 por 224                       |
| Xception            | 71          | 85          | 22.9                  | 299 por 299                       |
| Inception-ResNet-v2 | 164         | 209         | 55.9                  | 299 por 299                       |
| ShuffleNet          | 50          | 5.4         | 1.4                   | 224 por 224                       |
| NASNet-Mobile       |             | 20          | 5.3                   | 224 por 224                       |
| NASNet-Large        |             | 332         | 88.9                  | 331 por 331                       |

| Nombre          | Profundidad | Tamaño (MB) | Parámetros (millones) | Tamaño de entrada de las imágenes |
|-----------------|-------------|-------------|-----------------------|-----------------------------------|
| DarkNet-19      | 19          | 78          | 20.8                  | 256 por 256                       |
| DarkNet-53      | 53          | 155         | 41.6                  | 256 por 256                       |
| EfficientNet-b0 | 82          | 20          | 5.3                   | 224 por 224                       |
| AlexNet         | 8           | 227         | 61                    | 227 por 227                       |
| VGG-16          | 16          | 515         | 138                   | 224 por 224                       |
| VGG-19          | 19          | 535         | 144                   | 224 por 224                       |

Tabla 2. Tabla comparativa redes Deep Network Designer

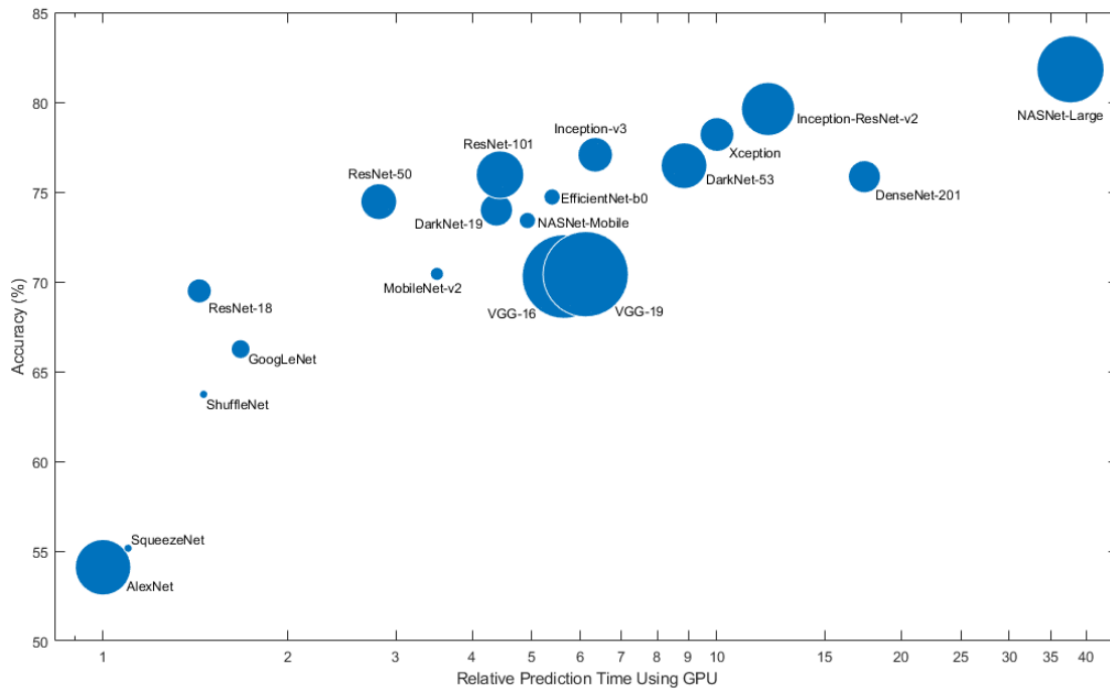
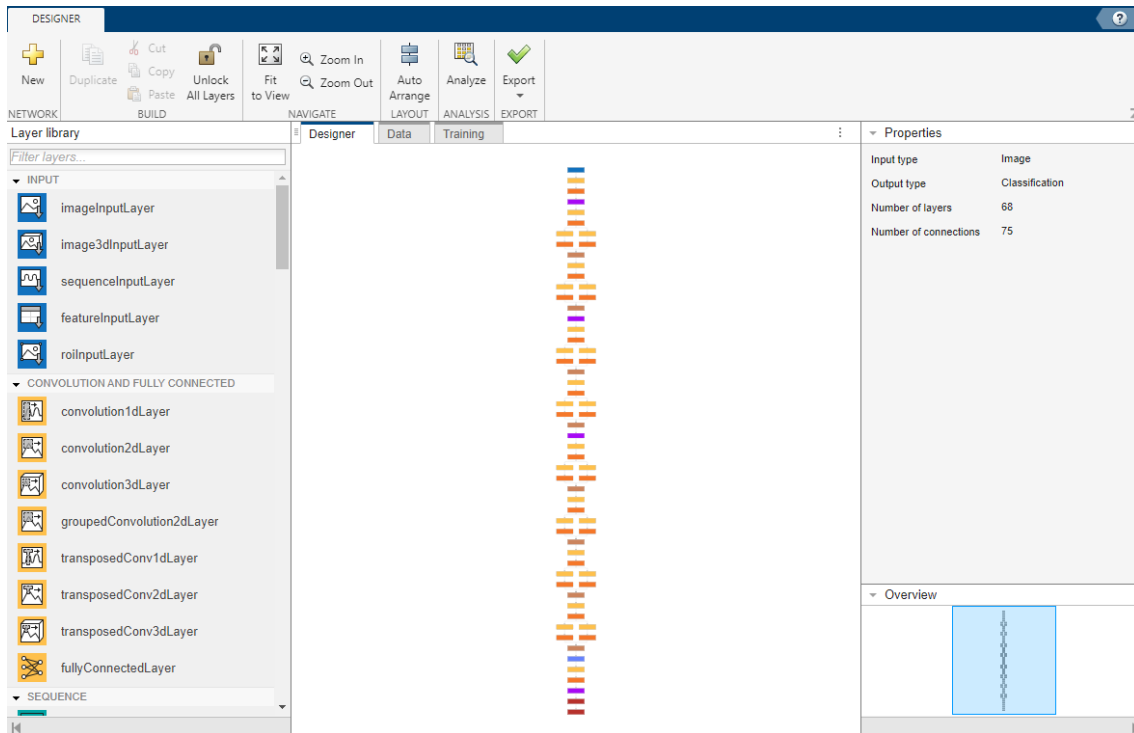


Figura 23. Gráfica comparativa redes Deep Network Designer

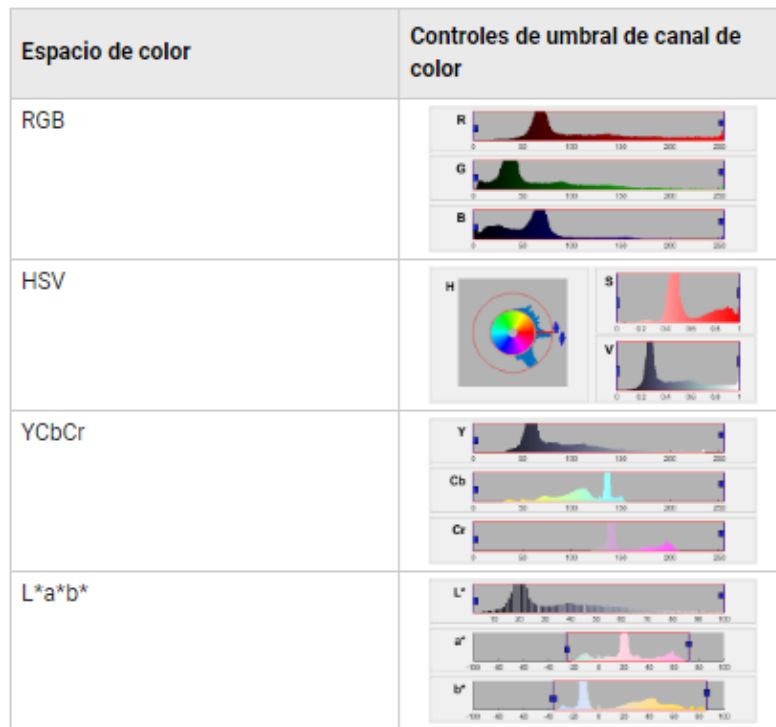
Según estas comparativas, se escogerán las redes que mejor se adapten a los requisitos del sistema y se probarán estas sobre la base de datos creadas para el entrenamiento y testeo de estas. Además, esta herramienta permite modificar cada una de las capas que forman la red lo que es muy importante de cara a adaptarla a este proyecto en particular. La figura 18 constata un ejemplo de la ventana de esta aplicación una vez seleccionada la red.



*Figura 24. Ventana principal ejemplificativa*

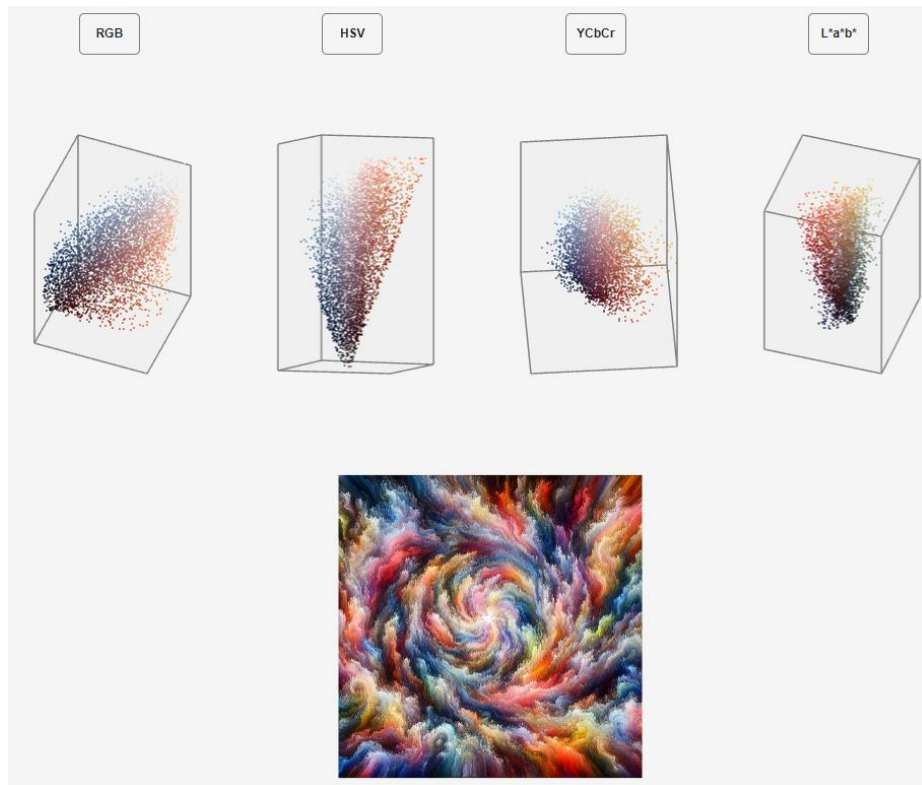
### 3.3.2. Color Thresholder

Esta herramienta de Matlab forma parte de la biblioteca Image Processing Toolbox [38], esta aplicación, permite segmentar una imagen dada según los colores que la forman. Esta segmentación, se podrá realizar mediante distintos espacios de colores según el objeto que se busque clasificar, a continuación, se adjunta los espacios de colores ofrecidos junto con un ejemplo de cada uno de ellos.

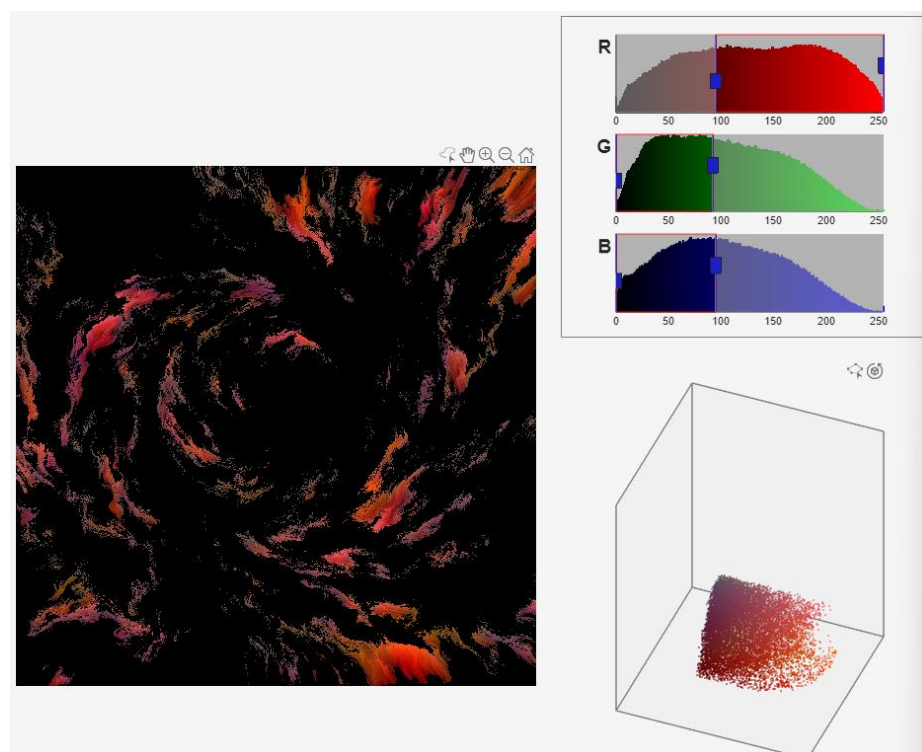


*Figura 25. Espacio de colores disponibles [39]*

Esta aplicación, será fundamental a la hora de hacer la detección del carril o la detección de la barra de las señales ya que se podrá segmentar la imagen de manera muy visual. Dependiendo de la imagen que se quiera clasificar, se seleccionará el espacio que más facilite la umbralización y una vez acabada esta se devolverá una función auxiliar que al ser llamada, realice la misma segmentación. A continuación, se adjuntan dos figuras ejemplificativas sobre el funcionamiento de esta aplicación.



*Figura 26. Ventana inicial para la segmentación por color*



*Figura 27. Segmentación según el espacio de color RGB*

### 3.4. Redes neuronales convolucionales (CNN)

Una red neuronal convolucional es un tipo de red neuronal artificial utilizada con el fin de extraer las características de una imagen para la posterior clasificación de imágenes similares. Para entrenarlas, es necesario aportarles una clasificación manual sobre la cual puedan aprender de una forma similar a la que lo haría el córtex visual del ser humano.

Para realizar esta clasificación, se realiza un procesamiento en cascada, es decir, primero identifica los rasgos más generales y básicos para posteriormente entrar en detalles más concretos. El nombre de estas redes viene del procedimiento que estas siguen ya que aplican un filtro, o kernel, convolucional a lo largo de los píxeles de la imagen para obtener el nuevo valor, su funcionamiento es el siguiente.

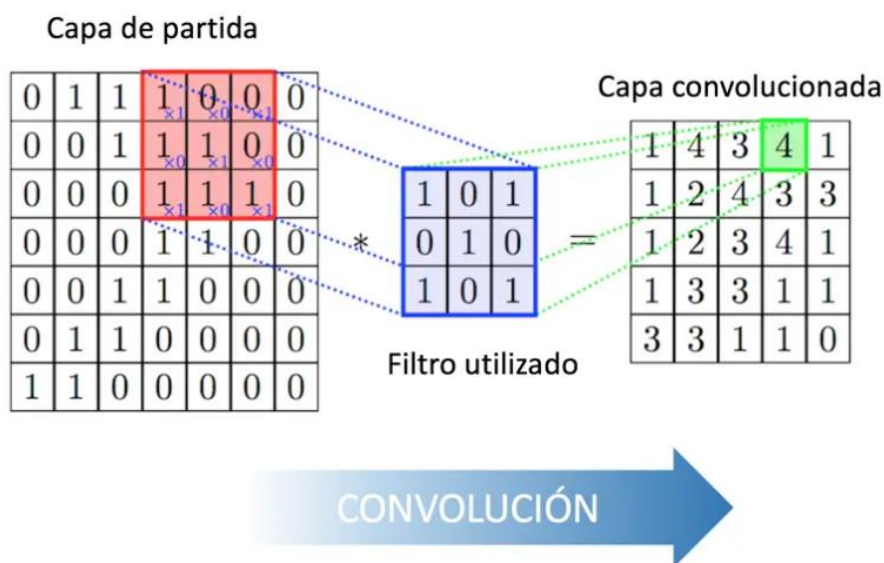


Figura 28. Explicación aplicación kernel convolucional [40]

Al aplicar este filtro a lo largo de la imagen se obtiene una imagen nueva que depende del filtro aplicado, por lo que cuanto más óptimo sea este, mejores resultados se obtendrán. En la siguiente figura, se muestra la aplicación de un kernel sobre una imagen inicial y la imagen resultante tras utilizar este.

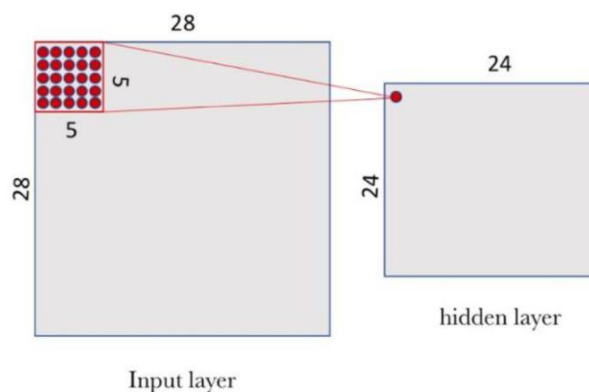


Figura 29. Aplicación de un filtro convolucional

Después de aplicar este filtro, tiene lugar un *polling* o reducción donde se obtendrán los rasgos más importante de la imagen y se desestimarán todas aquellas partes que no son importantes para la clasificación requerida. La imagen resultante, será un mapa de características de menor tamaño y que contenga solo las partes deseadas, en la imagen que se adjunta a continuación, se ha aplicado un kernel vertical para remarcar las líneas que sigan dicha orientación.

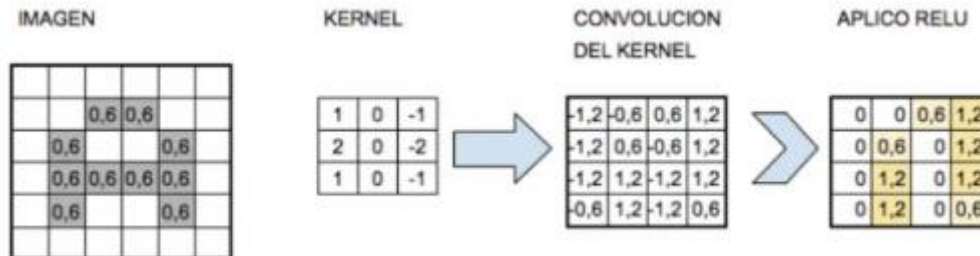


Figura 30. Obtención del mapa de características [41]

Al repetir este proceso en repetidas ocasiones, se obtiene una clasificación mucho más potente que la original y que permitirá al modelo clasificar imágenes nuevas con una precisión muy alta. La siguiente imagen, muestra las etapas que componen una red CNN de manera ejemplificativa, aunque esta arquitectura podrá variar ligeramente dependiendo de la red en particular.

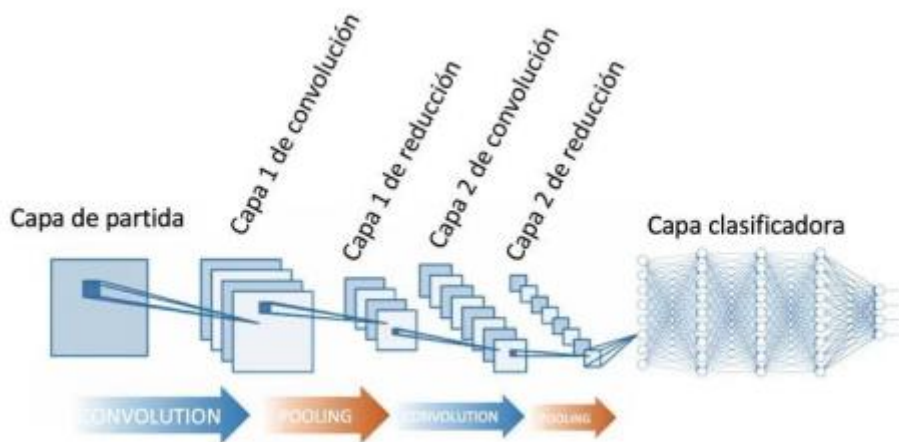


Figura 31. Red neuronal convolucional [40]

Aunque un solo kernel no sería capaz de detectar formas complejas, al aplicar este proceso de forma iterativa, se podrán detectar cada vez formas más complejas. Es por ello, que las CNN se representan en forma de embudo de tal forma que la imagen se va comprimiendo espacialmente hasta que se tienen todos los patrones de la imagen y una gran cantidad de mapas de características para realizar la clasificación. El funcionamiento de una CNN, de manera muy simplificada, sería como se muestra en la siguiente figura.

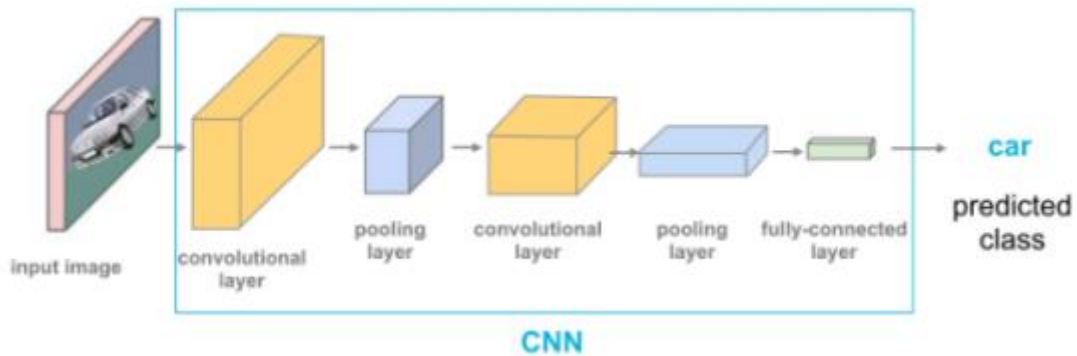


Figura 32. Arquitectura de una CNN [42]

### 3.5. Morfología matemática

La morfología matemática, es un conjunto amplio de operaciones que sirven para procesar imágenes basándose en distintas formas geométricas, el valor de cada píxel de salida se basa en la comparación de este píxel con sus vecinos. Se aplica exclusivamente sobre imágenes binarias o imágenes en escala de grises pero no tiene aplicación sobre imágenes en color [43].

Las dos operaciones principales de la morfología matemática son:

- **Dilatación (“imdilate”)**: El valor del píxel de salida es el máximo de los píxeles del entorno, sirve para hacer los objetos más visibles y rellenar posibles huecos.
- **Erosión (“imerode”)**: El valor del píxel de salida es el mínimo del entorno, sirve para limar objetos y asperezas.

La figura mostrada a continuación, ejemplifica una operación de dilatación sobre una imagen en escala de grises.

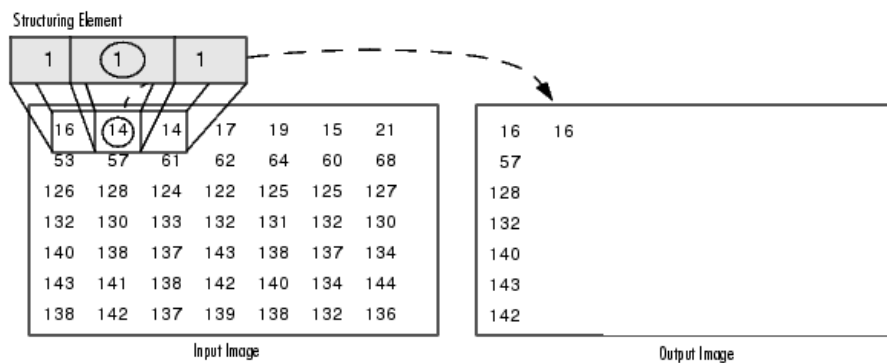


Figura 33. Aplicación sobre una imagen en escala de grises [43]

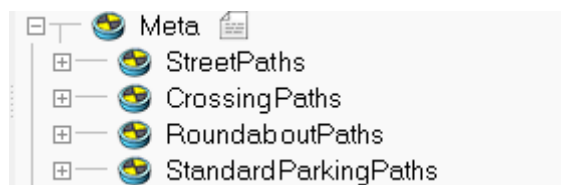
Estas técnicas, pueden combinarse para realizar una apertura (erosión+dilatación) si se desea eliminar posible ruido en la imagen de entrada o cierre (dilatación+erosión) si por lo contrario se busca obtener imágenes más robustas. Estas herramientas serán de gran utilidad para la segmentación del carril o la detección de la señal eliminando así posible ruido resultante de operaciones anteriores.



### 3.6. GPS y planificación de la trayectoria

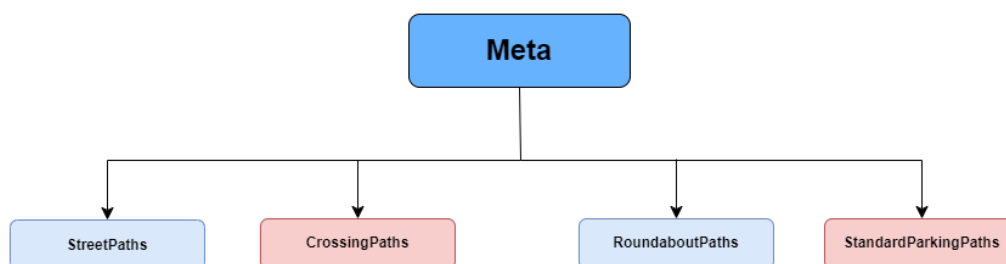
Como sistema alternativo a los planteados con anterioridad, se ha propuesto la implementación de un sistema de planificación de ruta por GPS mediante el cual el vehículo podrá circular de manera autónoma por la ruta más corta. Para ello, se han utilizado un conjunto de paths o caminos que recorren toda la escena, para posteriormente y mediante un algoritmo de control, seleccionar la ruta más corta para realizar este desplazamiento.

Este conjunto de paths de Coppelia, tanto los originarios del repositorio [19] como los creados para las partes modificadas que se explican en los capítulos siguientes, se almacenan en Coppelia Sim dentro de un objeto padre llamado Meta según la siguiente figura.



*Figura 34. Agrupación de las rutas en Coppelia*

Para clarificar la jerarquía de las rutas dentro de esta variable meta se adjunta el siguiente esquema.



*Figura 35. Esquema organizativo de las rutas*

Como se puede ver en las dos figuras anteriores, se clasifican las rutas según su tipo en calles, cruces, rotondas o parking, de esta forma la transmisión de los datos a Matlab se realiza de manera comprimida mediante un única línea de código. Posteriormente, se descomprimirá esta información ya en Matlab y se realizará la planificación de la trayectoria mediante el algoritmo de control creado con este propósito.

## 4. Detección de señales

A lo largo de este capítulo, se explica de manera detallada el algoritmo utilizado para la detección de las señales. Para ello, se divide este capítulo en distintos apartados donde se desarrollen en profundidad las distintas etapas que componen este algoritmo y que habilitarán el funcionamiento del vehículo en función de estas.

En ese trabajo, se ha decidido que el procesamiento de las señales se realice mediante redes neuronales convolucionales ya que, como se ha explicado en el capítulo anterior. Las

oportunidades que estas ofrecen en su escalabilidad y de cara al futuro son idóneas para un proyecto de estas características.

Además, aunque ya en el capítulo del estado del arte se explicó que las redes preentrenadas son mejores en este tipo de aplicaciones, se ha desarrollado una red neuronal propia para esta aplicación en exclusiva.

## 4.1. Creación de la Data base

Para poder entrenar una red neuronal, es necesario tener una base de datos con imágenes de cada tipo de objeto, en este caso señal, que se quiera analizar lo suficientemente larga y completa como para que represente de manera plena y fehaciente las situaciones donde se utilizará esta red. En este caso, se hará la clasificación sobre las imágenes que devuelve la cámara directamente sin segmentarlas o tratarlas, con ello, se desea lograr que la red entrenada sea capaz de adaptarse a todo tipo de señalización sin requerir de otras características o rasgos extras.

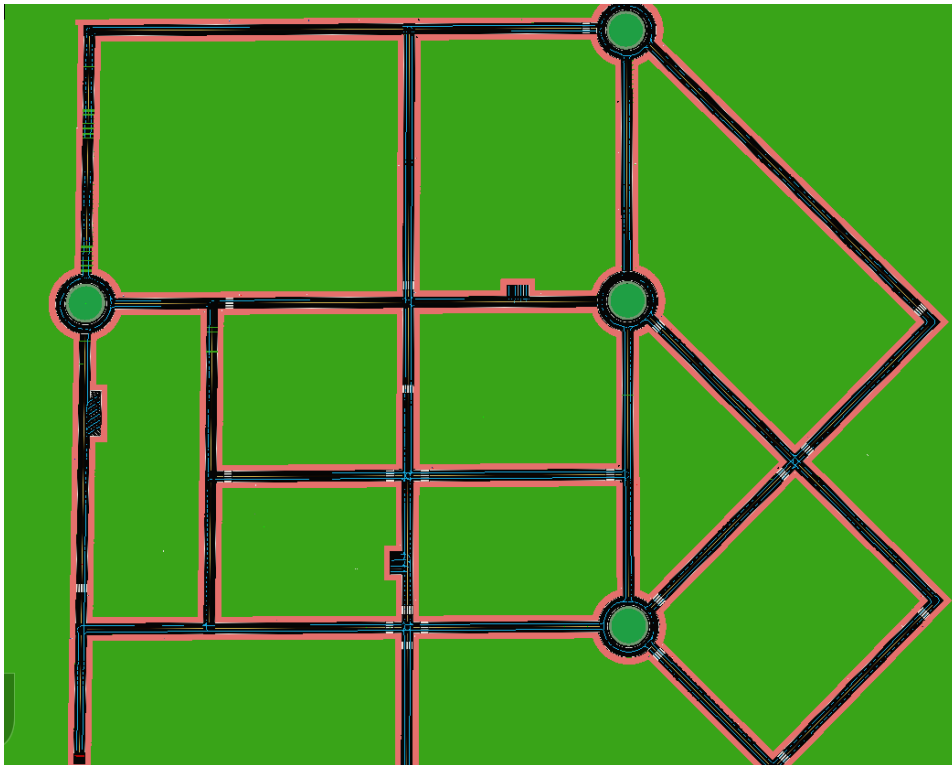
### 4.1.1. Escena Coppelia

Aunque el modelo final debe ser funcional sobre la escena que se ha explicado en el punto 3.1.3, para la obtención de las imágenes de la base de datos se ha decidido modificar esta ligeramente para mejorar la calidad de la base de datos.

Las dos modificaciones principales que se han realizado son:

- **Eliminación de edificios:** En la escena utilizada para la obtención de la base de datos, se han eliminado los edificios que aparecen en el entorno. Esto se debe, a que al ser estos muy grandes en comparación con las señales, podrían hacer al modelo incurrir en falsos positivos y que la red no detecte la señal sino su entorno.
- **Modificación de las señales:** Por el motivo mencionado en el primer punto, se han movido las señales a posiciones diferentes a lo largo de la obtención de las imágenes. Además, se ha comprobado que cada imagen de la base de datos solo contenga una señal ya que, con esta red neuronal se querrá detectar las señales de manera clara e individual. Posteriormente, y en caso de detectar varias señales simultáneamente durante la ejecución, se seleccionará la más cercana al vehículo mediante el tratamiento de la imagen por las funciones auxiliares que se verán en los siguientes apartados.

Una vez hechas estas modificaciones, la escena de entrenamiento desde el plano cenital se vería de la siguiente forma.



*Figura 36. Plano cenital de la escena entrenamiento*

### 4.1.2. Obtención de las imágenes

Este tipo de redes, requieren bases de datos muy grandes, en torno a 600 imágenes por categoría aunque esta cifra varía dependiendo de la dificultad de la clasificación esperada, y teniendo en cuenta que este entorno tiene 9 señales distintas, esto supondría una enorme cantidad de tiempo guardando y etiquetando imágenes. Para agilizar este proceso, se ha creado un script en el cual se seleccionan las características bajo las cuales se desea que se ejecute la simulación (número de imágenes a capturar y orientación y traslación de la cámara) y se clasifican estas en la categoría pertinente.

A continuación, se muestran el fragmento del script donde se inicia la simulación aunque esta parte del código será necesaria siempre que se requiera una conexión entre Matlab y Coppelia.

```
% Inicio simulación
client = RemoteAPIClient();
sim = client.getObject('sim');
fprintf('Program started\n');
client.setStepping(true);
sim.startSimulation();

client.step();

startTime = sim.getSimulationTime();
t = startTime;
```

*Figura 37. Código para iniciar la simulación*

La siguiente figura, se corresponde con el fragmento de código donde se definen las características bajo las cuales tiene lugar la simulación, remarcar que estos son ejemplificativos y variarán dependiendo del objetivo de la simulación.

```
% Ángulos iniciales sobre los que aplicar variación (en grados)
angulos_iniciales = [86.389, 0.01, -0.383];

% Número de imágenes a capturar
inicio_img = 0;
num_imagenes = inicio_img + 100;

% Rango de variación de orientación de la cámara
rango_orientacion = 5;

% Rango de variación de posición en los ejes x e y
rango_posicion_x = 0.3;
rango_posicion_y = 0.3;
capturas_realizadas = inicio_img; % Contador de imágenes capturadas
```

**Figura 38. Configuración características simulación**

Cabe destacar, que aunque en el entorno real no habrá oscilaciones ni baches, con el fin de enriquecer la red y el sistema, se añaden rotaciones aleatorias a la toma de imágenes. De esta forma, si la red es capaz de detectar y analizar las señales en estas circunstancias límites, en unas condiciones normales de ejecución funcionará de manera impecable.

La captura de las imágenes desde Coppelia a Matlab se realiza mediante la función `getImage` que se adjunta a continuación, donde `sim` es el objeto de la simulación definido al inicio y `handle` el sensor de visión que se utiliza para tomar estas. Mediante esta función se obtiene la imagen captada y se adapta a las características necesarias para trabajar con ella en el futuro.

```
function img = getImage(sim, handle)
    [img, resX, resY] = sim.getVisionSensorCharImage(handle);
    img = flip(permute(reshape(img, 3, resX, resY), [3 2 1]), 1);
end
```

**Figura 39. Función `getImage`**

El conjunto de características definidas en la figura 38, y que definen los rangos de variabilidad entre los que se modificará el vehículo en cada iteración, se aplican mediante las siguientes líneas de código.

```
while t - startTime < 20 && capturas_realizadas < num_imagenes
% Generar y aplicar orientaciones y posiciones aleatorias
for i = inicio_img:num_imagenes
    client.step();
    visionSensorHandle = sim.getObject('/ViewCamera');

    % Capturar imagen
    img = getImage(sim, visionSensorHandle);

    % Generar ángulos aleatorios para los ejes X, Y y Z
    angulos_aleatorios = randi([-rango_orientacion, rango_orientacion], 1, 3);

    % Valor entre -rango_posicion_x y rango_posicion_x
    posicion_x_aleatoria = (2*rand - 1) * rango_posicion_x;
    % Valor entre -rango_posicion_y y rango_posicion_y
    posicion_y_aleatoria = (2*rand - 1) * rango_posicion_y;

    % Aplicar variación a los ángulos iniciales
    angulos_nuevos = angulos_iniciales + angulos_aleatorios;

    % Enviar comando para establecer la orientación del sensor
    sim.setObjectOrientation(visionSensorHandle, -1, angulos_nuevos*pi/180);

    % Obtener la posición actual del sensor
    posicion_actual = sim.getObjectPosition(visionSensorHandle, -1);

    % Acceder al contenido de la celda para obtener el vector numérico
    posicion_actual = cell2mat(posicion_actual);

    % Aplicar variación a la posición en los ejes X e Y
    nueva_posicion_x = posicion_actual(1) + posicion_x_aleatoria;
    nueva_posicion_y = posicion_actual(2) + posicion_y_aleatoria;

    % Establecer la nueva posición del sensor
    nueva_posicion = [nueva_posicion_x, nueva_posicion_y, posicion_actual(3)];
    sim.setObjectPosition(visionSensorHandle, -1, nueva_posicion);

    % Reorganizar y guardar la imagen en formato PNG
    nombre_archivo = fullfile('...../Images\Stop', sprintf('Stop_%d.png', capturas_realizadas + 1));
    imwrite(img, nombre_archivo);

    disp(['Imagen ', num2str(capturas_realizadas + 1), ' capturada y guardada']);

    capturas_realizadas = capturas_realizadas + 1;
end
t = sim.getSimulationTime();
end
```

*Figura 40. Código principal para hacer la Data base*

Una vez se tienen todas las imágenes etiquetadas en sus respectivas categorías, se podrá pasar a la siguiente etapa, preparar las imágenes de esta base de datos para el posterior entreno de una red neuronal, el código completo se encuentra de manera íntegra en el anexo 1.1.

## 4.2. Preentrenamiento

Antes de entrenar la red neuronal, es importante preparar las imágenes para optimizar el entrenamiento de esta. En este caso, las etapas de preentrenamiento son el etiquetado o *labeling*, el redimensionado o *resizing* y la división de las imágenes o *spliting* en las carpetas de entrenamiento, validación y test.

### 4.2.1. Labeling










Esta etapa, sirve para clasificar las imágenes según la señal que deba ser detectada en ellas, otorgando a cada una de ellas la identificación que deberá ser corroborada por la red en el entrenamiento posterior.

En este modelo, la etapa de etiquetado está integrada en el código del anexo 1.1 ya que de esta manera el proceso será más eficiente y no se tendrá que revisar y etiquetar las cerca de 6000 imágenes manualmente. La parte del código que realiza esta función se adjunta a continuación:

```
% Reorganizar y guardar la imagen en formato PNG
nombre_archivo = fullfile(' ../../../../Images\Stop', sprintf('Stop_%d.png', capturas_realizadas + 1));
imwrite(img, nombre_archivo);
```

*Figura 41. Etiquetado automático de las imágenes*

Una vez capturadas todas las imágenes, estas quedarán ordenadas en el directorio local de la siguiente forma:

|   |                     |                  |                     |
|---|---------------------|------------------|---------------------|
|    | Callejon            | 02/04/2024 22:32 | Carpeta de archivos |
|   | Limite velocidad 30 | 02/04/2024 22:33 | Carpeta de archivos |
|  | Parking             | 02/04/2024 22:33 | Carpeta de archivos |
|  | Paso de peatones    | 02/04/2024 22:33 | Carpeta de archivos |
|  | Prohibido           | 02/04/2024 22:33 | Carpeta de archivos |
|  | Rotonda             | 02/04/2024 22:33 | Carpeta de archivos |
|  | Semaforo            | 02/04/2024 22:34 | Carpeta de archivos |
|  | Sentido único       | 02/04/2024 22:34 | Carpeta de archivos |
|  | Stop                | 02/04/2024 22:34 | Carpeta de archivos |

*Figura 42. Data base ordenada y etiquetada*

### 4.2.2. Resizing

Como ya se mencionó en la tabla 2, cada red neuronal requiere un tamaño de las imágenes de entrada concreto y por lo tanto las imágenes de nuestra base de datos debe ser redimensionada para adaptarse a la características necesarias de cada red.

Al igual que en el apartado anterior, este *resizing* se realizará de manera automática mediante un script de Matlab. De esta forma, se logra redimensionar, y agrupar en una nueva carpeta, las imágenes de nuestra base de datos mediante una sola ejecución, el código completo utilizado para realizar esta tarea se encuentra en el anexo 1.2.

Más en particular, la función que realiza el redimensionado de las imágenes es la función *imresize*, la cual recibe como parámetros de entrada la imagen original y las dimensiones deseadas

de la imagen final, en la siguiente figura se muestra un ejemplo de como se usaría esta función para obtener una imagen de salida de 224x224.

```
% Redimensionar las imágenes
k = imresize(im, [224, 224]);
```

*Figura 43. Comando imresize*

### 4.2.3. Splitting

Una vez todas las imágenes están ya en el formato correcto, el último paso será repartir estas imágenes en tres categorías distintas según su función (entrenamiento, validación y test). Las imágenes de entrenamiento serán las utilizadas para entrenar la red y deben ser mayoría, en este caso el 70% de la base de datos, aunque este valor podrá variar ligeramente dependiendo de la aplicación.

La parte de validación (un 20% en este proyecto), también se usa para el entrenamiento de la red aunque solo para optimizar la red mediante los hiperparámetros, que se reajustarán durante el entrenamiento del modelo a medida que este tiene lugar.

La última parte de las imágenes, las de test (un 10% en este caso), no se tiene en cuenta para entrenar la red sino que se utilizarán para comprobar la eficiencia de la red y los resultados que esta devuelve con un conjunto de imágenes que no ha trabajado con anterioridad.

Este *split* se puede realizar de diferentes formas, e incluso la propia herramienta de entrenamiento de la red permite hacerlo, pero en este caso se hará mediante un script de Matlab que las divide de manera aleatoria y mediante los porcentajes deseados, en este caso 70-10-10.

Lo primero que se hace, al igual que en los demás casos, es proporcionar las carpetas en las cuales se encuentran las imágenes que se deben repartir, lo que se realiza mediante el siguiente fragmento de código.

```
folderName = '../Images resized/Images sin edificios';
srcDir = fullfile(folderName);
srcSubfolders = dir(fullfile(srcDir, '**'));

% Eliminar carpetas que no sean subcarpetas
srcSubfolders = srcSubfolders([srcSubfolders.isdir] & ~strcmp({srcSubfolders.name}, '.') & ~strcmp({srcSubfolders.name}, '..'));
```

*Figura 44. Dirección carpetas para el Split*

Posteriormente, el reparto se hará asignando una cierta cantidad de imágenes, según el porcentaje requerido, a cada carpeta hasta haber repartido todas las imágenes. El indexado de las imágenes y el reparto de estas se hace mediante el código mostrado en las dos siguientes figuras.

```
% Obtener los índices de división
[numTrain, numVal, numTest] = dividerand(numImages, trainRatio, valRatio, testRatio);
```

*Figura 45. Obtención de los índices de las imágenes*

```
% Copiar las imágenes a las carpetas correspondientes
for k = 1:length(numTrain)
    imgName = srcFiles(k).name;
    imgPath = fullfile(subfolderPath, imgName);
    copyfile(imgPath, fullfile(trainFolder, imgName));
end

for k = (length(numTrain)+1):(length(numTrain) + length(numVal))
    imgName = srcFiles(k).name;
    imgPath = fullfile(subfolderPath, imgName);
    copyfile(imgPath, fullfile(valFolder, imgName));
end

for k = (length(numTrain) + length(numVal)+1):numImages
    imgName = srcFiles(k).name;
    imgPath = fullfile(subfolderPath, imgName);
    copyfile(imgPath, fullfile(testFolder, imgName));
end
```

*Figura 46. Bucles para el Split de las imágenes de la base de datos*

El código completo se encuentra en el anexo 1.3.

### 4.3. Soluciones propuestas

Para cumplir esta funcionalidad, se han propuesto tres redes distintas, con dos arquitecturas distintas, para posteriormente comparar los resultados y elegir la mejor opción para este modelo. La primera arquitectura planteada será la detección de las señales con las imágenes ya recortadas lo que aunque facilita enormemente el proceso de detección aunque dificulta el procesamiento de la imagen general para la obtención de este recorte. La red que se basa en esta arquitectura se expone detalladamente en el siguiente capítulo.

#### 4.3.1. SqueezeNet

La primera red entrenada es la red SqueezeNet, esta, destaca por su gran precisión en relación con su consumo de CPU (figura 22), esto será especialmente útil en este proyecto ya que se buscará obtener no solo una opción eficaz sino que sea eficiente.

Para este entrenamiento, se ha utilizado el conjunto de imágenes facilitado por los creadores de la escena [19] en el cual las imágenes de las señales están recortadas para que solo se observe la señal en sí, la siguiente figura, ejemplifica una de estas imágenes una vez ya ha sido redimensionada con el código del anexo 1.2.





Figura 47. Imagen entrenamiento SqueezeNet

La principal ventaja de esta arquitectura es la facilidad mediante la cual la red será capaz de detectar señales una vez las imágenes de entrada han sido recortadas lo que teniendo en cuenta la profundidad de la red será determinante.

### 4.3.1.1. Entrenamiento

El entrenamiento de esta red neuronal convolucional se ha realizado mediante la herramienta de Matlab Deep Network Designer que se explicó en el punto 3.3.1, para ello, se han modificado las capas finales de la red para que esta tenga en cuenta 10 grupos distintos y se le ha subido el peso a estas para que sean más determinantes.

Estas modificaciones se han realizado en la última capa convolucional y en la capa de clasificación, la arquitectura resultante de esta es la siguiente:

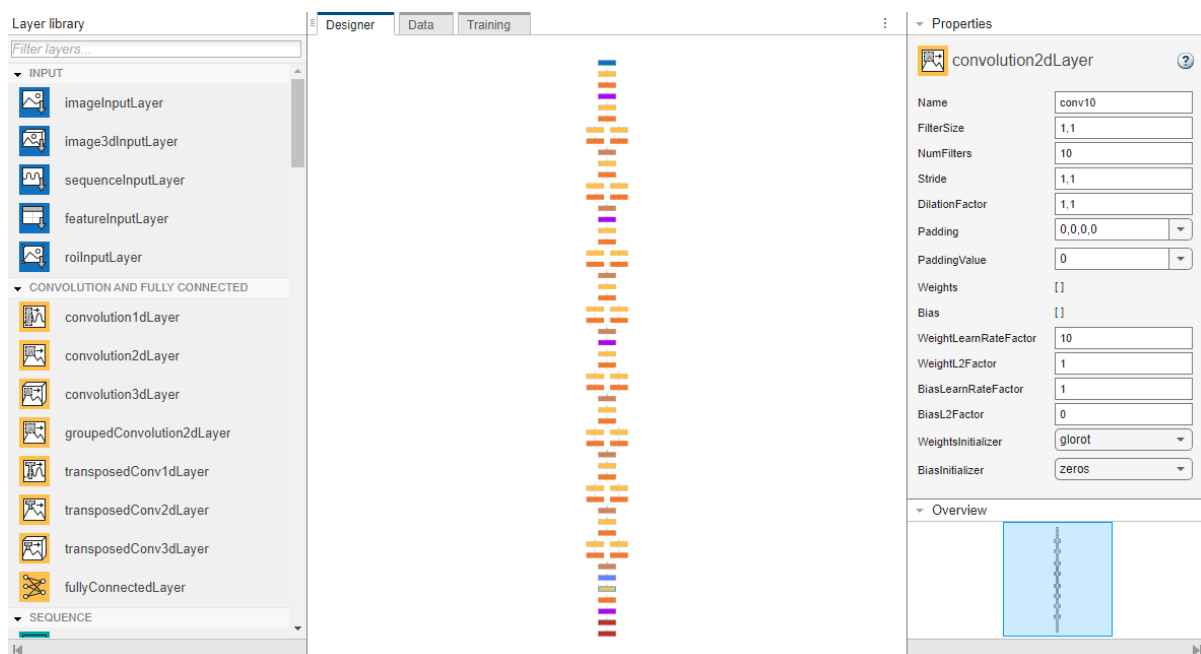
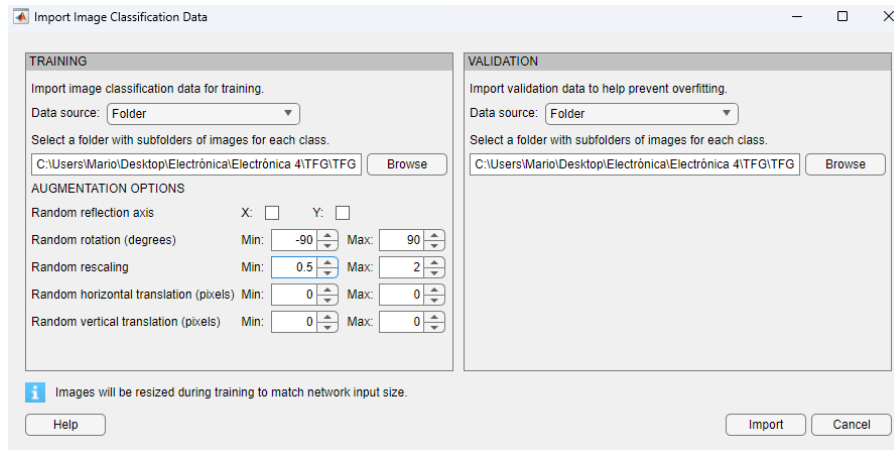


Figura 48. Arquitectura red SqueezeNet

Los datos que se usarán para este entrenamiento serán los de entrenamiento y validación ya que los de test se reservarán para probar la red una vez esta ya ha sido entrenada.



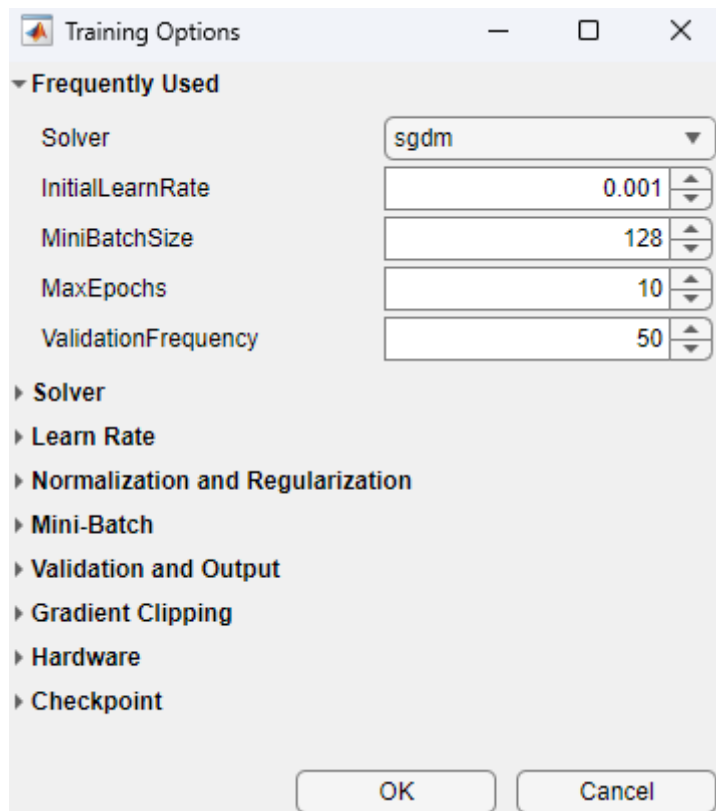
**Figura 49. Opciones para importar la base de datos**

Como se puede observar en la figura anterior, las imágenes que se utilizarán son rotadas y reescaladas para enriquecer aún más la red, una vez aplicados estos ajustes los datos de entrenamiento se muestran a continuación.



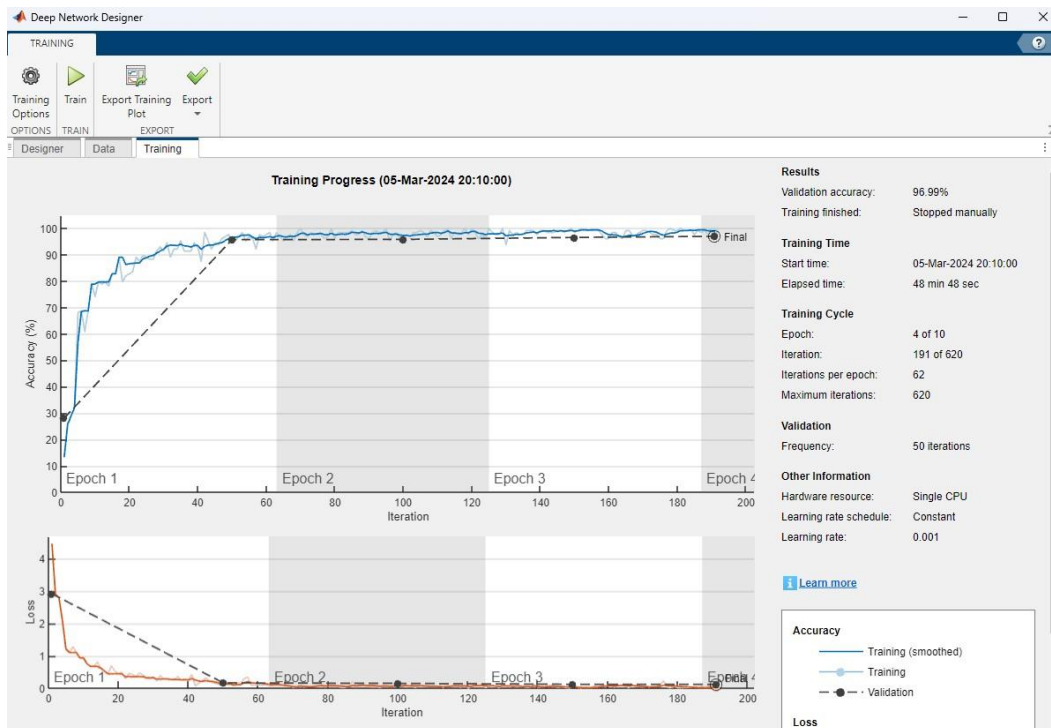
**Figura 50. Training Data para SqueezeNet**

Las opciones para realizar el entrenamiento de la red con estas imágenes se muestran a continuación.



*Figura 51. Opciones entrenamiento SqueezeNet*

Tras entrenar la red durante un breve periodo de tiempo, menos de 49 minutos en este caso lo que es muy poco para una red de este tipo, se obtienen los siguientes resultados.



*Figura 52. Resultados entrenamiento SqueezeNet*

Como se puede observar, los resultados son muy buenos ya que se alcanza el 97% de precisión sobre las imágenes de validación lo que podría rozar la perfección mediante la utilización algoritmos de control que eliminen los falsos positivos.

### 4.3.1.2. Resultados (Test)

Aunque los resultados han sido buenos en la fase de entrenamiento, se han creado dos scripts adicionales para probar el funcionamiento de esta con las imágenes que se reservaron para el test. El primero de ellos, recorrerá todas las imágenes del test y utilizará con cada una de ellas la red entrenada para mostrar en una figura los aciertos y fallos resultantes.

En las siguientes figuras se adjunta el procedimiento para obtener las imágenes requeridas, procedimiento análogo al utilizado en el script de preentrenamiento, junto con la creación de una carpeta para almacenar todas aquellas detecciones fallidas de cara a posibles análisis manuales posteriores.

```
folderName = '../..\\Images resized\\Images sin edificios\\Test';  
srcDir = fullfile(folderName);  
srcSubfolders = dir(fullfile(srcDir, '*'));
```

*Figura 53. Dirección de las imágenes del test*

```
% Crear una carpeta para las detecciones fallidas  
outputFolder = fullfile(srcDir, '../detecciones_fallidas');  
if ~isfolder(outputFolder)  
    mkdir(outputFolder);  
end
```

*Figura 54. Dirección carpeta para los fallos*

Además, se ha creado una función auxiliar para asignar un color a cada señal con el fin de mejorar el muestreo de los datos finales, esta función se adjunta a continuación.

```

%% Función para clasificar cada imagen por el objeto detectado
function color= seleccion_colores(tipo)
    switch tipo
        case 'Callejon'
            color = 'r'; % Rojo
        case 'Limite velocidad 30'
            color = 'g'; % Verde
        case 'Parking'
            color = 'b'; % Azul
        case 'Paso de peatones'
            color = 'c'; % Cian
        case 'Prohibido'
            color = 'm'; % Magenta
        case 'Reversed'
            color = 'y'; % Amarillo
        case 'Rotonda'
            color = 'k'; % Negro
        case 'Semáforo'
            color = [0.5 0.5 0.5]; % Gris
        case 'Sentido único'
            color = [1 0.5 0]; % Naranja
        case 'Stop'
            color = [0 1 1]; % Turquesa
        otherwise
            color = 'k'; % Por defecto, negro
    end

```

Figura 55. Función selección de colores

Los resultados de aplicar este script, que se encuentra íntegro en el anexo 1.4, se adjuntan a continuación:

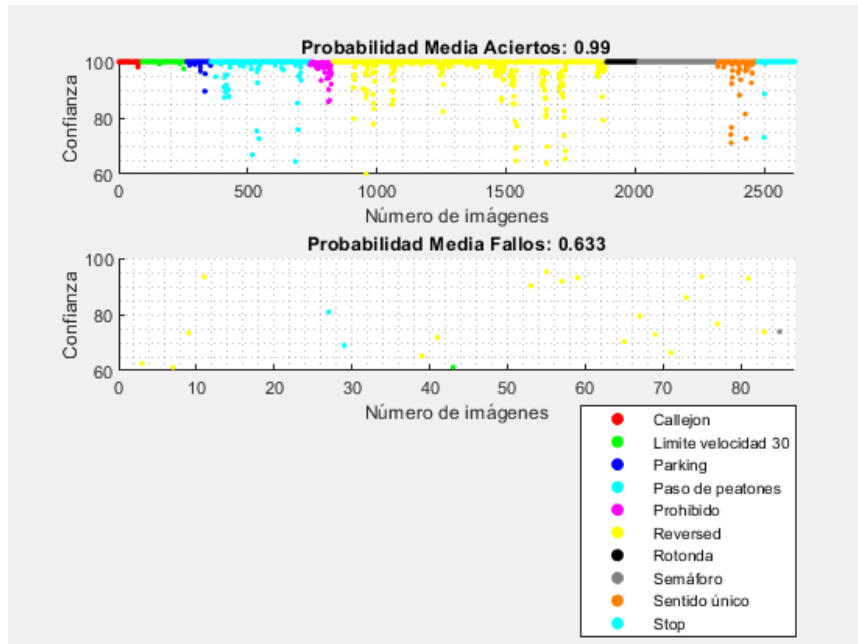


Figura 56. Gráfica resultados SqueezeNet

```
Resultados:
Número de detecciones correctas:2619
Número de errores de detección:43
Porcentaje de acierto:98.3847%
Certidumbre en detecciones correctas:0.99045
Certidumbre en errores de detección:0.63336
Figuras con las probabilidades de detección y el valor medio generadas.
```

*Figura 57. Resultados SqueezeNet*

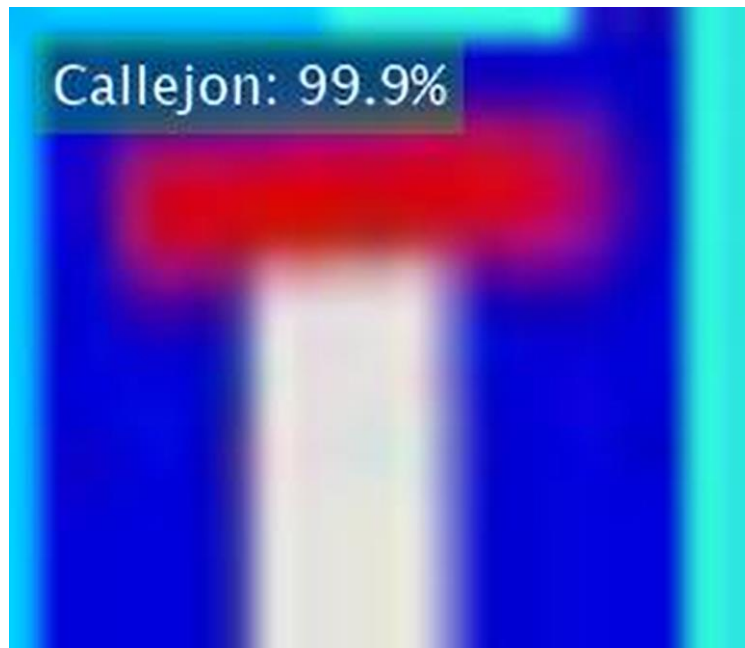
Como se puede observar, los resultados de esta red neuronal son muy buenos y coinciden con los que se habían obtenido en el entrenamiento por lo que se puede deducir que este fue bueno y que la red neuronal funciona correctamente.

Sin embargo, se ha creado otro script para demostrar de forma gráfica estos resultados, este, genera un video donde se analiza cada imagen del test y se añade al fotograma la detección realizada y la confianza en la detección. Esto permite observar de forma gráfica los resultados de este test en tiempo real y optimizar el modelo en caso de ser necesario, el código utilizado para escribir sobre las imágenes es el siguiente:

```
% Mostrar el título en el video
titleText = string(label) + ": " + num2str(100*max(probs),3) + "%";
I = insertText(I, [10 10], titleText, 'FontSize', 16, 'BoxColor', 'green', 'BoxOpacity', 0.3, 'TextColor', 'white');
```

*Figura 58. Escritura en la imagen*

A continuación, se adjunta un frame del video que se obtiene al realizar esta prueba, el código completo de este script se puede encontrar en el anexo 1.5.



*Figura 59. Fotograma ejemplificativo del video creado por el script de test*

## 4.3.2. Resnet50

Aunque se ha utilizado la misma herramienta en ambos casos, Deep Network Designer, en este caso se plantea una arquitectura distinta ya que en vez de utilizar las imágenes ya recortadas que se ofrecía en el repositorio de git [19], se utilizarán las herramientas mencionadas en el punto 4.1.

Es por ello, que como en este caso la detección de señales será mucho más compleja, ya que los píxeles de la señal representan un porcentaje muy pequeño de la imagen total. Es por ello, que se ha decidido optar por una red más potente, lo que se puede comprobar observando la gráfica comparativa de la figura 23.

### 4.3.2.1. Entrenamiento

Además de las técnicas mencionadas al principio de este capítulo, serán necesarias ciertas modificaciones, como ya se mencionó con la SqueezeNet, para adaptar la arquitectura de la CNN al caso particular de este modelo.

La modificación que se realiza sobre las capas de esta red, tiene como fin ajustar el filtro de salida a 9 categorías (ya que en este caso no se añadirán las imágenes reversed por su falta de aplicabilidad), y el peso de la última capa convolucional para que esta sea 10 veces más determinante.

La siguiente figura, muestra una visión general de la arquitectura de la red Resnet50 junto con la capa que ha sido modificada por los motivos mencionados anteriormente.

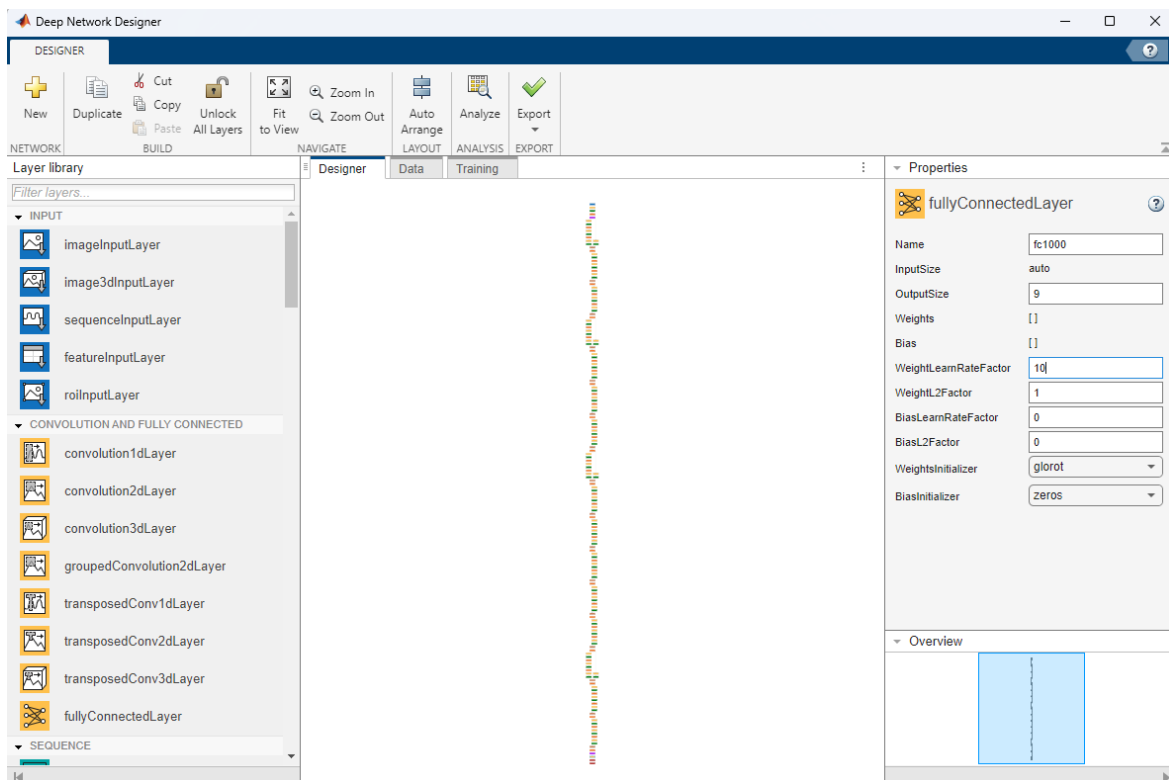
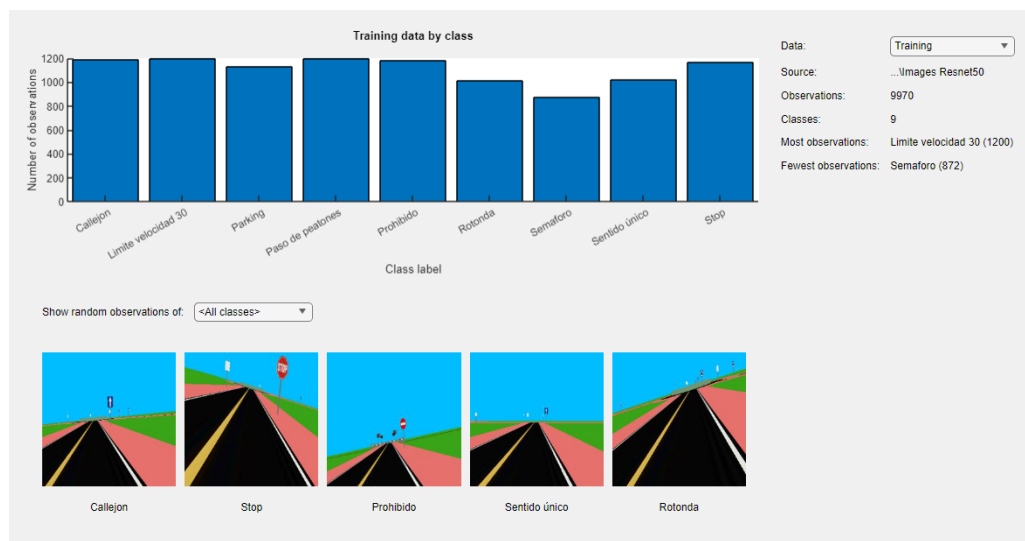


Figura 60. Arquitectura modificada Resnet50

Como se puede observar a simple vista, esta red cuenta con muchas más capas que el modelo anterior lo que hará que el proceso de entrenamiento sea mucho más largo. El proceso de importación de los datos de entrenamiento para esta red se realiza de manera análoga al proceso realizado con la red SqueezeNet, solo que en este caso, con la base de datos creada para este objetivo en particular.

Sin embargo, debido a que en este caso ya se realizaron las transformaciones y rotaciones en la captura de las imágenes realizar estas cuando se importen las imágenes será redundante. A continuación, se adjuntan los datos de entrenamiento ya importados para el entrenamiento de esta red.



**Figura 61. Data de entrenamiento Resnet50**

Cabe destacar, que en este caso el número de imágenes de cada señal, incluso sin utilizar las *reversed*, será mayor en esta aplicación ya que al ser más complejo el procedimiento, serán necesarias más muestras, las opciones de entrenamiento se observan en la siguiente figura.



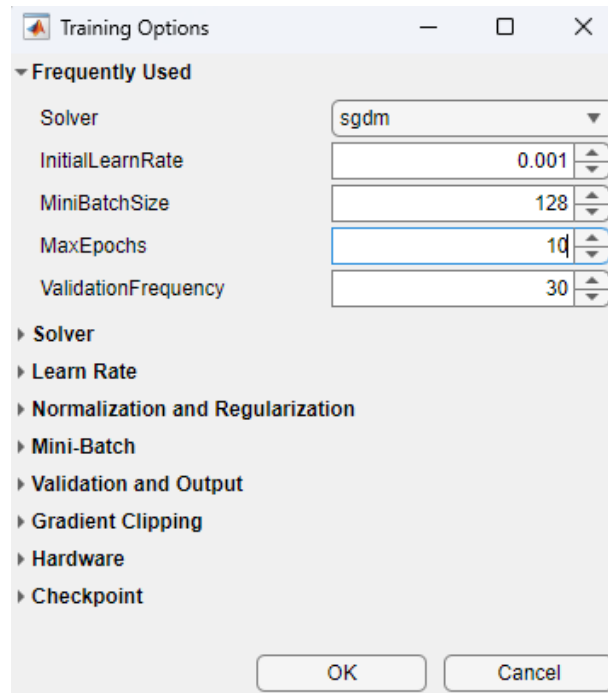


Figura 62. Opciones entrenamiento Resnet50

Tras realizar el entrenamiento de esta red neuronal, con las opciones configuradas en la figura anterior, la gráfica resultante del entrenamiento de esta red durante casi 5 horas es la siguiente.

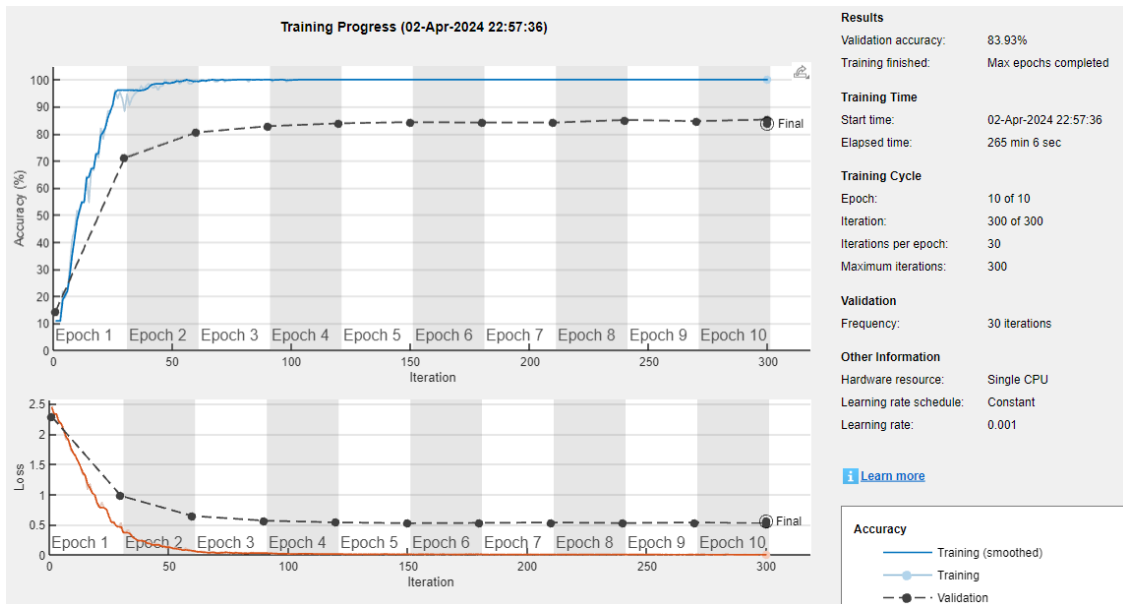


Figura 63. Gráfica del entrenamiento de Resnet50

Como se puede observar, el porcentaje de acierto final sobre los datos de validación será ligeramente inferior, 83.93%, y aunque se dejase más tiempo entrenando, este no mejoraría ya que a partir de la tercera época se produce overfitting.

### 4.3.2.2. Resultados

Utilizando los scripts ya explicados durante el punto 4.3.1.2, donde se exponían los comandos, funciones y scripts creados para comprobar la validez de esta red sobre las imágenes de test. A continuación, se adjuntan las figuras resultantes de aplicar el script del punto de anexos 1.4 con la red neuronal convolucional Resnet50.

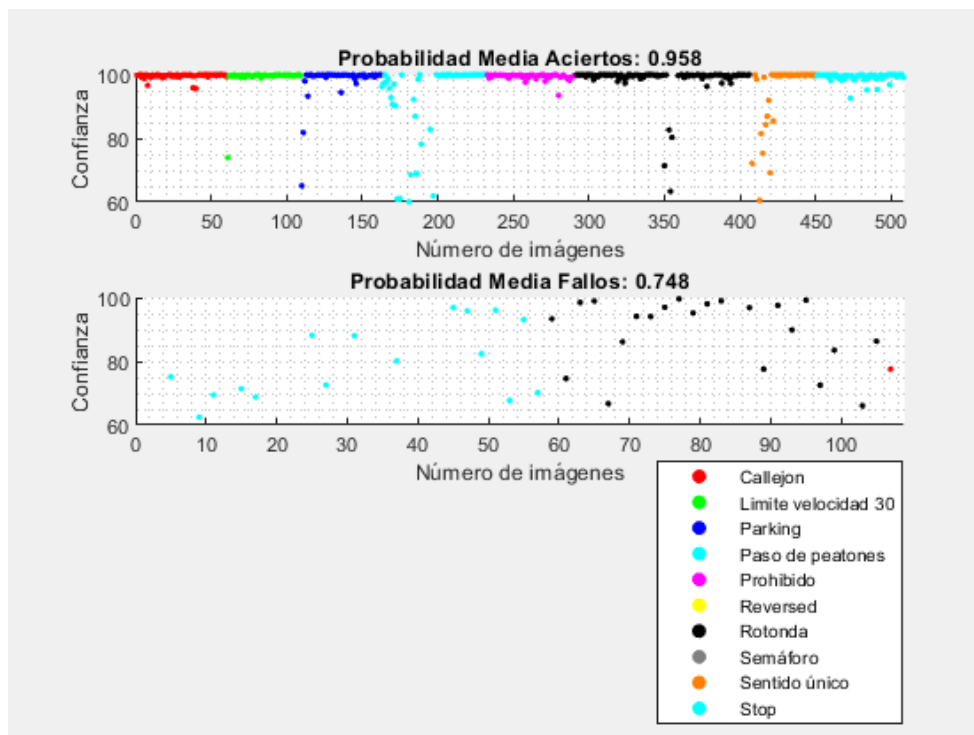


Figura 64. Gráfica aciertos y fallos red Resnet50

```

Resultados:
Número de detecciones correctas:508
Número de errores de detección:54
Porcentaje de acierto:90.3915%
Certidumbre en detecciones correctas:0.95843
Certidumbre en errores de detección:0.74768
    
```

Figura 65. Resultados Resnet50 sobre las imágenes de test

Como se puede observar, los resultados de esta red también son muy positivos aunque ligeramente inferiores, por los motivos expuestos con anterioridad.

Sin embargo, esto no será limitante ya que como la figura 65 muestra, los falsos positivos son muy pocos, entorno al 10%, lo que sumado a algoritmos de decisión podrían reducir estos al mínimo. Finalmente, se adjunta un frame ejemplificativo del video generado con esta red neuronal y el código que se encuentra en anexo 1.5.



Figura 66. Frame ejemplificativo video Resnet50

### 4.3.3. YOLOv2

En este caso en particular, la red neuronal convolucional se ha entrenado sin la herramienta de Deep Network Designer ya que al hacerlo por código se podrá configurar la red de una manera más sencilla. Esta red se basa en la arquitectura de las redes YOLOv2 [44] ya que es un tipo de red muy buena en este tipo de entornos y que ya se ha utilizado en vehículos autónomos para la detección de vehículos en tiempo real, propósito similar al de este sistema [45].

Además, esta red es de código abierto y destaca por su gran velocidad lo que es muy importante para esta aplicación en particular. Esta red, basa su reconocimiento en la técnica de los *bounding boxes*, esta técnica consiste en la clasificación de la imagen en distintos cuadrados para conocer cuales contienen información importante y cuales son desechables.

A continuación se adjunta una imagen aclaratoria sobre el reconocimiento mediante bounding boxes que plantean las redes YOLO.

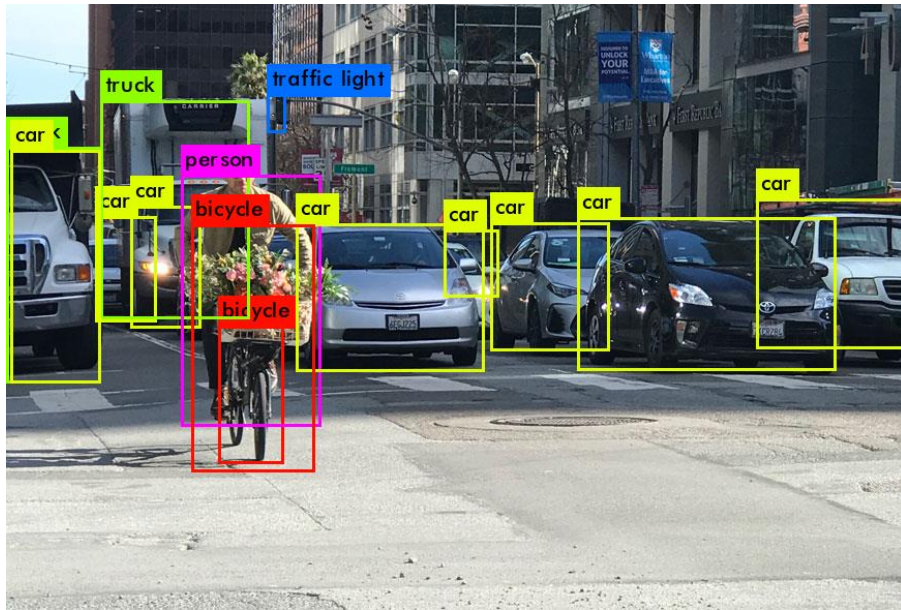


Figura 67. Detección en una imagen por una red de arquitectura YOLO [46]

Aunque la inserción manual de estos bounding boxes no es estrictamente necesaria, de hecho en este proyecto no se aplican por el enorme coste que supondría en comparación con otras redes, si que podrían suponer una mejora sustancial en la clasificación.

Esta red, es ideal para el reconocimiento en entornos abiertos de objetos muy concretos por lo que se adapta perfectamente al propósito de este proyecto como se ha podido observar en la figura anterior. La arquitectura de esta red convolucional es la que se adjunta a continuación, donde se puede observar cada capa por separado y como esta interviene en la red.

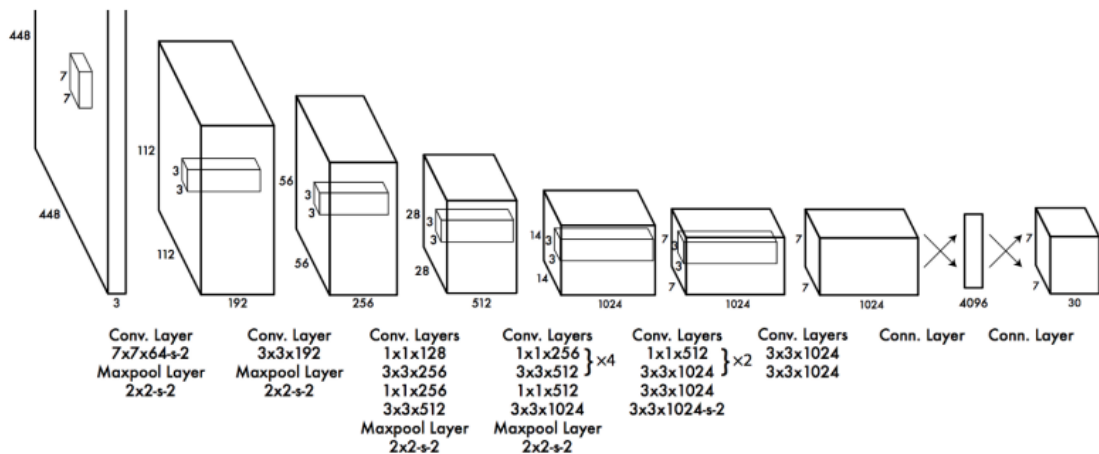


Figura 68. Arquitectura red YOLOv2 [47]

### 4.3.3.1. Entrenamiento

Para esta red, el tamaño de las entradas será de 416x416 por lo que primeramente se aplicarán los scripts explicados en preentrenamiento para la preparación de estas imágenes para el entrenamiento de la red.

Una vez estas imágenes han sido redimensionadas y divididas en las respectivas subcarpetas, se comienza con el entrenamiento de esta red, aunque el código completo puede encontrarse en el anexo 1.6, en este subapartado se explicarán las partes más resaltables de este script.

Los parámetros de las capas de la red que se han utilizado para este modelo se adjuntan en la siguiente figura, cabe destacar que se ha añadido una parte del código para reducir el overfitting ya que este es un problema típico de este tipo de redes más simples [48]. El overfitting, es el proceso de sobreentrenamiento de un algoritmo de aprendizaje y resulta en un error en el entrenamiento mediante la cual la red alcanza un valor de 100% de precisión para el entrenamiento pero baja para los demás tipos [49].

```
% Red Propia
numClasses = numel(categories(imds_train.Labels));
layers = [
    imageInputLayer(input_size)

    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,16,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];

% Agrega regularización L2
weight_decay = 0.001;
layers(2).Weights = layers(2).Weights / (1 + weight_decay);
layers(6).Weights = layers(6).Weights / (1 + weight_decay);
layers(10).Weights = layers(10).Weights / (1 + weight_decay);
```

**Figura 69. Capas del entrenamiento de la red YOLO**

El overfitting mencionado con anterioridad se reduce mediante la inclusión de las siguientes líneas de código.

```
% Agrega capas de dropout para reducir el overfitting
layers(4) = dropoutLayer(0.3);
layers(8) = dropoutLayer(0.3);
```

**Figura 70. Reducción del overfitting**

Las opciones de entrenamiento bajo la cual se desarrolla este modelo son similares a la de los casos anteriores y se configuran por código mediante las siguientes líneas de código.

```
options = trainingOptions('sgdm', ...
    'MinibatchSize', 10, ...
    'MaxEpochs', 25, ...
    'ValidationData', validationData, ...
    'ValidationFrequency', 150, ...
    'InitialLearnRate', 1e-4, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropPeriod', 20, ...
    'LearnRateDropFactor', 0.2, ...
    'Plots', 'training-progress');
```

Figura 71. Opciones de entrenamiento de la red YOLO

### 4.3.3.2. Resultados

Para analizar los resultados de esta red, se han utilizado los mismos scripts y técnicas que en las demás redes ya que de esta forma se podrá comparar perfectamente los resultados de cada una de ellas y tomar la mejor decisión en consecuencia. Los resultados del entrenamiento que se han obtenido con la configuración de la figura 71 son los siguientes.

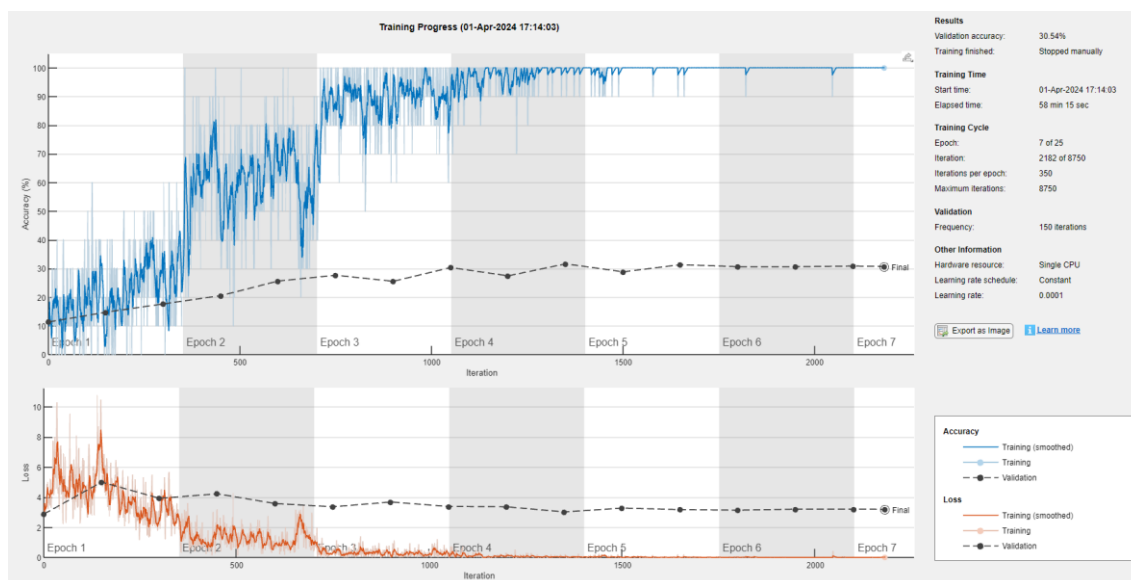
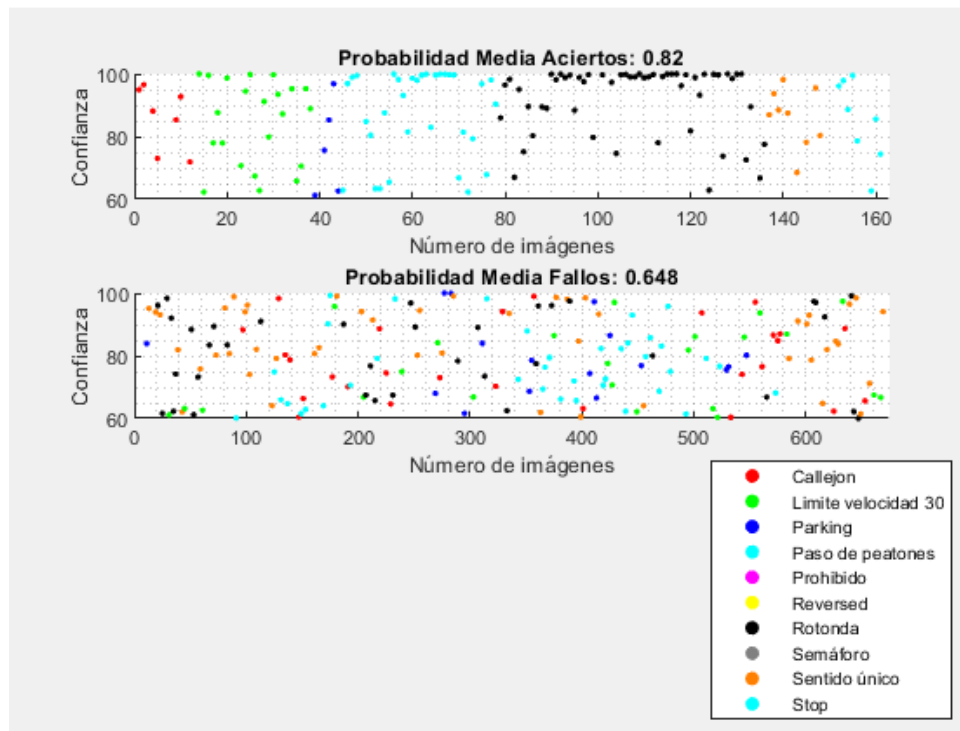


Figura 72. Resultados entrenamiento red YOLOv2

Como se puede observar, los resultados obtenidos con esta red para la validación son muy malos, un 30% de precisión para los datos de validación. Además, a pesar de las medidas aplicadas para reducir el overfitting como aumentar la data base, añadir el dropout, etcétera, serán insuficientes para eliminarlo completamente. Por lo tanto, no se podrá obtener una mayor precisión por muchas épocas que se entrene la red, siendo cada época un repaso completo a las imágenes de entrenamiento.

En las siguientes figuras se muestran los resultados de aplicar el script del anexo 1.4 con la red neuronal YOLO.



*Figura 73. Gráfica aciertos y fallos red YOLOv2*

```

Resultados:
Número de detecciones correctas:162
Número de errores de detección:337
Porcentaje de acierto:32.4649%
Certidumbre en detecciones correctas:0.82049
Certidumbre en errores de detección:0.64847
Figuras con las probabilidades de detección y el valor medio generadas.
    
```

*Figura 74. Resultados precisión red YOLOv2*

Los resultados obtenidos en este script, muestran correlación con los de la figura 72 como cabría esperar y aunque la probabilidad media de los aciertos es relativamente superior a la de los fallos, 0.82 contra 0.648, la precisión de esta red no será suficiente para esta aplicación.

## 4.4. Selección modelo

Teniendo en cuenta todo lo mencionado en los capítulos anteriores, se ha decidido que la red que mejor se acopla a lo que se busca en este proyecto es la red Resnet50. Esto se debe, a que aunque la red SqueezeNet ofrecía una mayor precisión y tasa de acierto, la arquitectura de la red seleccionada es mucho mejor de cara a su escalabilidad y adaptabilidad ya que con esta red no será necesario analizar imágenes recortadas, ya que este recorte en ocasiones puede ser extremadamente complejo. Por los motivos mencionados, en el capítulo 4.3.3.2 la red YOLO se descartará para esta implementación debido a su incapacidad para obtener un buen porcentaje de detecciones correctas.

Además, estas características son fundamentales en los vehículos autónomos ya que al ser un campo tan novedoso y donde el entorno es muy incierto, ser adaptativo es un rasgo diferencial. A

su vez, la red seleccionada cuenta con una mayor cantidad de parámetros de entrada, lo que hará que esta mejore con mayor velocidad a medida que aumente su uso.

A continuación, se adjunta un tabla comparativa para explicar simplificadaamente la toma de esta decisión teniendo en cuenta las características de cada una de las redes.

| Red        | % acierto | Polivalencia | Parámetros (millones) | Funcionalidad |
|------------|-----------|--------------|-----------------------|---------------|
| SqueezeNet | 98.38     | Baja         | 1.4                   | Plena         |
| Resnet50   | 90.3915   | Muy alta     | 25.6                  | Plena         |
| YOLOv2     | 32.4649   | Muy alta     | 0.005                 | Mala          |

Tabla 3. Comparación redes neuronales entrenadas

## 4.5. Implementación del logaritmo de detección

Una vez se ha decidido la red neuronal que se utilizará en el modelo, se procede a la creación de un algoritmo para su utilización en el control del vehículo para mejorar aún más la precisión de esta. A continuación, se adjunta un esquema general para comprender mejor los inputs y outputs que tiene la función auxiliar encargada de esto.

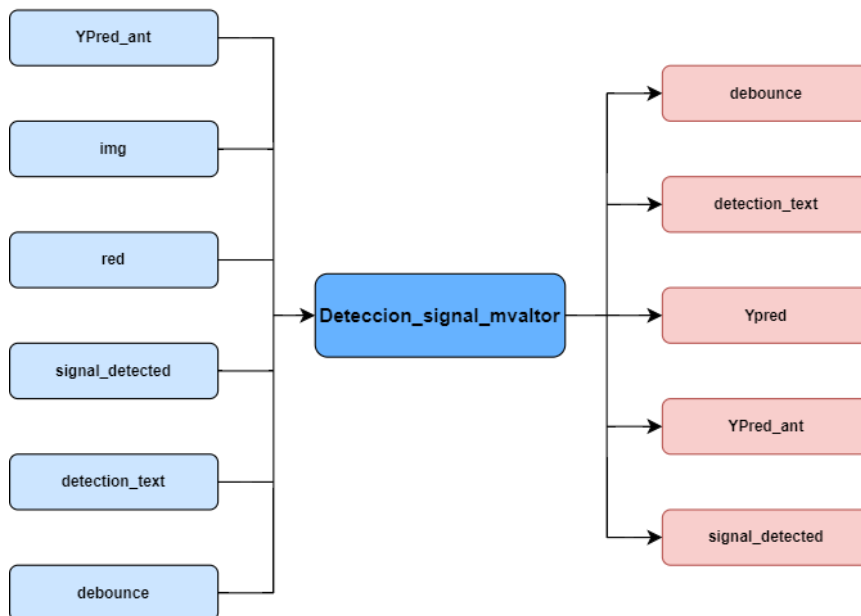


Figura 75. Esquema cabecera de la función para la detección de señales

En esta función, lo primero que se hará es redimensionar la imagen que se está capturando para poder analizarlas mediante la función classify [50] y con la red neuronal que ha sido entrenada con este objetivo con el código que se adjunta a continuación.



```

% Dimensiones de la imagen
[x, y, z] = size(img);
% Redimensionamos la imagen si es necesario
if any([x, y, z] ~= [224, 224, 3])
    img = imresize(img, [224, 224]);
end

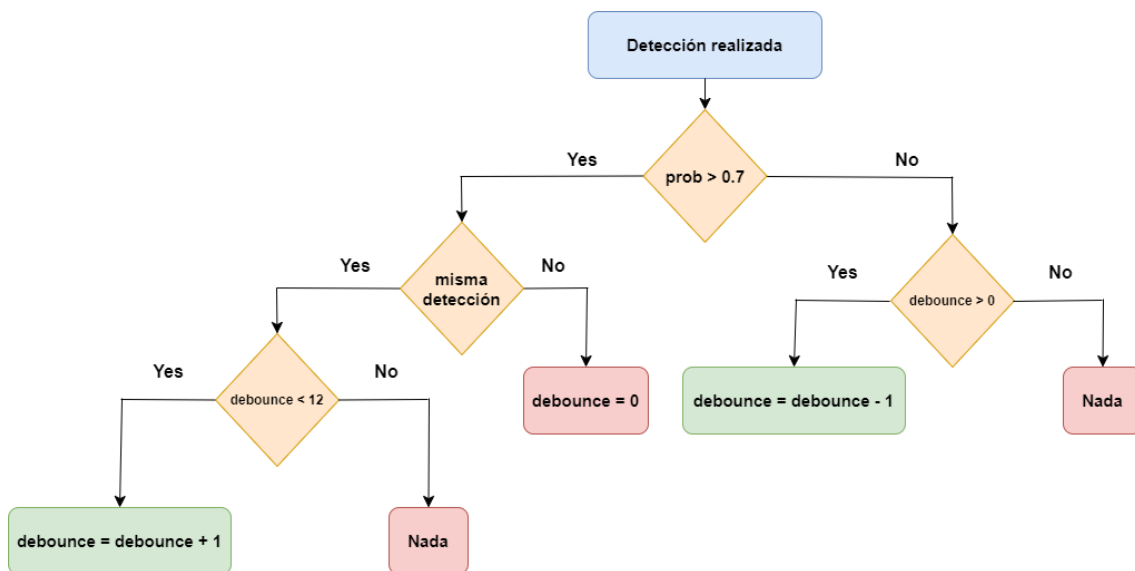
% Analizamos la imagen
[YPred,probs] = classify(red.Resnet50_mvalor, img);
    
```

*Figura 76. Análisis de la imagen por la CNN*

Posteriormente, y basándose en los datos devueltos durante el testeo de la red neuronal que se pueden observar en la figura 64, se crea un filtro mediante el cual solo se considera como válidas para la detección aquellas con más de 0.7 de probabilidad.

Además, en caso de que esta detección sea igual que la anterior se sumará a la variable “debounce” 1, hasta que esta valga 12, mientras que si hay un cambio en la detección se reseteará esta variable. De esta forma, se logra que sean necesarias varias detecciones consecutivas de la misma señal, además, esta variable tendrá un valor máximo de 12 para que la duración de la detección no tenga un peso excesivo sobre las futuras actuaciones.

A continuación, se muestra un esquema con la lógica que se sigue dentro de esta función para esta primera parte de detección.



*Figura 77. Esquema lógico para la detección*

La aplicación de este esquema mediante código se muestra en las siguientes figuras.

```
% Se analiza la imagen
[YPred,probs] = classify(red.Resnet50_mvaltor, img);

cambio_signal = 0.7;
prob_max = max(probs);
```

*Figura 78. Clasificación de la imagen por la red neuronal*

```
% cambio detección y filtrado
if(prob_max >= cambio_signal)
    if (YPred_ant == YPred)
        if debounce < 12
            debounce = debounce + 1;
        end
    else
        %Cambio en la detección
        debounce = 0;
    end
else
    %Falta de certeza en la detección
    if debounce > 0
        debounce = debounce - 1;
    end
end
```

*Figura 79. Secuencia lógica para el procesamiento de la detección*

Posteriormente y en caso de que haya habido 6 detecciones seguidas con una probabilidad mayor de 0.7, siendo la señal detectada la misma en todas ellas, se activará la variable booleana `signal_detected` a `true`, para activar la actuación respectiva de la señal, y se prepara el texto que se mostrará por pantalla.

Sin embargo, en caso de que esto no haya ocurrido la variable `signal_detected` se pondrá a `false` y se continuará con la ejecución, de esta forma, se logra aumentar la robustez del algoritmo una vez se ha detectado una señal dificultando el cambio de detección a no ser que sea muy claro. Con ello, se busca evitar los falsos positivos que puedan aparecer a lo largo de la ejecución ya que estos podrían provocar una conducción errática o accidentes derivados de esta detección. En la siguiente figura se muestra el código para la implementación de esta última parte del algoritmo de detección.

```
if (debounce >= 6 & YPred_ant == YPred)
    YPred_ant = YPred;
    % La señal si ha pasado el control
    detection_text = char(YPred) + ": " + num2str(max(probs)*100, '%.2f') + "%";
    signal_detected = true;
else
    YPred_ant = YPred;
    % La señal no ha pasado el control
    YPred = "No se detectó ninguna señal con suficiente certeza.";
    detection_text = char(YPred);
    signal_detected = false;
end
```

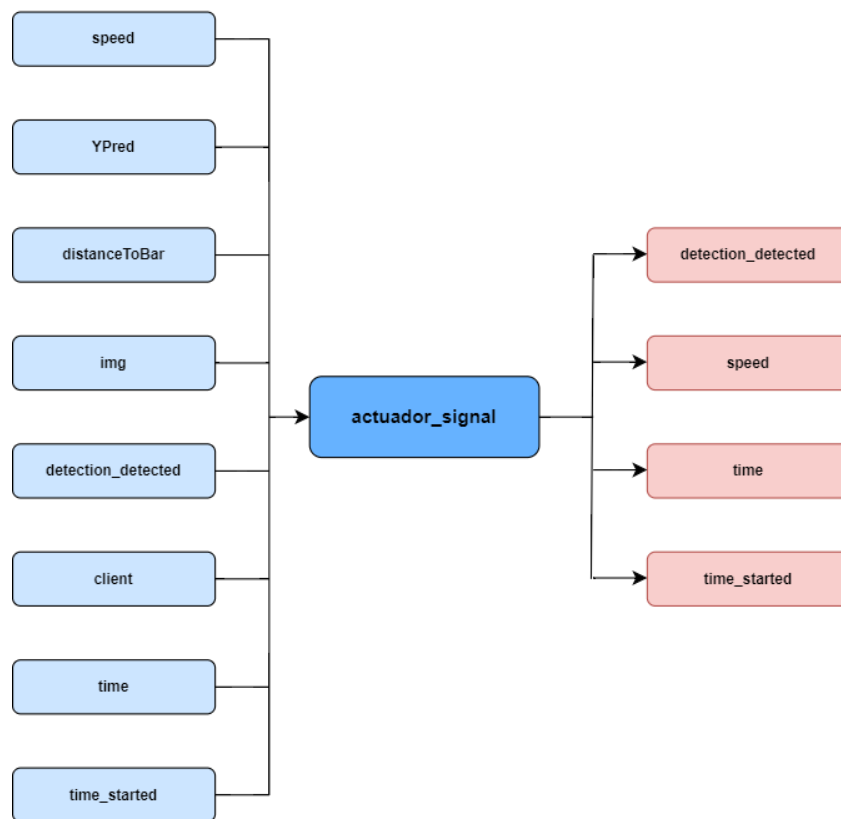
*Figura 80. Implementación de la última parte del algoritmo de detección de señales*

El código completo de esta función se encuentra de manera íntegra en el anexo 1.7.

## 4.6. Respuesta a las señales

Con el fin de aplicar la funcionalidad de esta parte del algoritmo sobre la simulación, se han creado algunos algoritmos de control específicos para cuando se detecten estas señales. Es importante recalcar, que en la realidad los vehículos cuentan con una gran cantidad de sensores que permiten al vehículo tomar decisiones con una mayor certeza por lo que a lo largo de esta simulación se usarán estos algoritmos de manera orientativa.

El esquema de las entradas y salidas de la función auxiliar utilizada para coordinar la respuesta del vehículo ante las distintas señales se adjunta a continuación.



*Figura 81. Esquema variable actuador para las señales*

A continuación, se adjunta el esquema de la implementación de esta función mediante un switch con todos los posibles casos que se podrían dar, cada una de las señales analizadas y el no haber detectado ninguna señal.

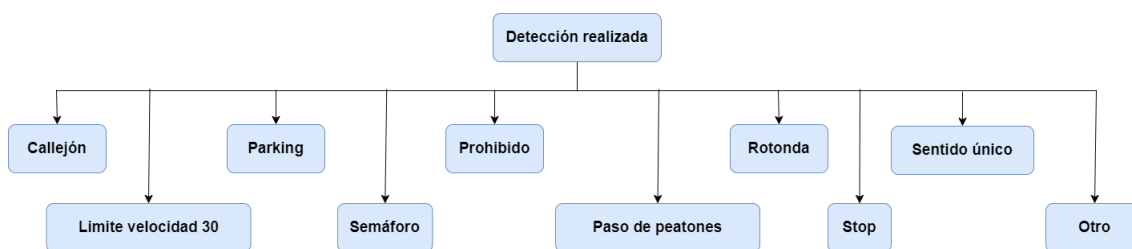


Figura 82. Árbol de decisión según la detección realizada

Según el tipo de actuación que se requiera en cada situación, se han creado 3 grupos (velocidad estándar, frenado o parada) que se explican en los siguientes subapartados aunque el código completo de esta función auxiliar se encuentra en el anexo 1.8.

### 4.6.1. Velocidad estándar

En este grupo, se encuentran aquellas señales que no intervienen en la velocidad estándar y que simplemente devuelven la velocidad a la que debería circular el vehículo, pudiendo ser esta la nominal o no. Aunque todas estas casuísticas podrían agruparse en un único caso, se ha dividido en cada señal por separado para facilitar futuras ampliaciones o aplicaciones en otro tipo de entornos.

Dentro de este grupo se encuentran las señales del límite de velocidad a 30, el sentido único y el no haber detectado ninguna señal, la implementación de estos 3 casos se adjunta a continuación.

```

case 'Sentido único'
    detection_detected = false;
    speed = 1.5;

case 'Limite velocidad 30'
    speed = 0.3;
    detection_detected = false;

otherwise
    detection_detected = false;
    speed = 1.5;
  
```

Figura 83. Implementación sentido único, 30 y nada

### 4.6.2. Frenado

Dentro de esta categoría, se encuentran aquellas señales que no necesariamente requieren que el vehículo frene aunque será altamente recomendable para la precisión y/o seguridad de las futuras tareas. En esta, se encuentran las señales de rotonda y parking cuya implementación dentro de esta función auxiliar es la siguiente.

```
case 'Parking'  
    detection_detected = false;  
    speed = frenado(distanceToBar);  
  
case 'Rotonda'  
    detection_detected = false;  
    speed = frenado(distanceToBar);
```

*Figura 84. Implementación parking y rotonda*

Como se puede observar, este proceso de frenado se realiza mediante otra función auxiliar dependiente de la distancia a la señal, y por lo tanto, al lugar que requiere esta reducción de velocidad. Esta función irá reduciendo gradualmente la velocidad hasta un mínimo de 0.3 cuando la distancia sea menor de 0.6 metros, a continuación, se muestra el contenido de dicha función.

```
function speed = frenado(distanceToBar)  
    if distanceToBar < 0.6  
        speed = 0.3;  
    elseif distanceToBar < 1.2  
        speed = 0.6;  
    else  
        speed = 1.5;  
    end  
end
```

*Figura 85. Función auxiliar para el frenado*

De esta forma, se logrará, además de una reducción de velocidad más paulatina, incrementar la velocidad del vehículo en los tramos rectos optimizando así el tiempo en el que el vehículo complete la ruta deseada.

### 4.6.3. Parada

El tercer tipo, está formado por todas aquellas señales en las cuales se acaba deteniendo el vehículo, dentro de esta clasificación se encuentran las señales de callejón, prohibido, semáforo y stop. A continuación, se adjunta la clasificación de estos casos dentro de la función de actuación de señales.

```
case 'Prohibido'  
    detection_detected = false;  
    speed = frenado_to_0(distanceToBar);  
  
case 'Callejon'  
    speed = frenado_to_0(distanceToBar);  
    detection_detected = false;  
  
case 'Semaforo'  
    detection_detected = false;  
    type = detector_luz(img);  
    if strcmp(type, 'rojo')  
        speed = 0;  
    elseif strcmp(type, 'ambar')  
        speed = 0.6;  
    elseif strcmp(type, 'verde')  
        speed = 1.5;  
    end  
  
case 'Stop'  
    [detection_detected, speed, time, time_started] = detencion_stop (client, detection_detected, distanceToBar, time, time_started);
```

*Figura 86. Implementación prohibido, stop, semáforo y callejón*

Como se puede observar, para hacer estas detenciones se utilizarán 3 funciones auxiliares frenado\_to\_0 (Prohibido y Callejón), detector\_luz (Semáforo) y detención\_stop (Stop) por lo que se explicará cada una de estas por separado en los siguientes apartados.

#### 4.6.3.1. Señales de prohibido y callejón

Para estas funciones, se utilizará la función externa frenado\_to\_0, esta función es similar a descrita en la figura 85 solo que en este caso la velocidad final será 0 en lugar de 0.3, a continuación, se adjunta esta función.

```
function speed = frenado_to_0(distanceToBar)
    if distanceToBar < 0.6
        speed = 0;
    elseif distanceToBar < 1.2
        speed = 0.6;
    else
        speed = 1.5;
    end
end
```

Figura 87. Implementación función frenado\_to\_0

Como se puede ver, esta función es análoga a la utilizada para reducir la velocidad y hará que el vehículo se detenga delante de la señal detectada.

#### 4.6.3.2. Señal de Stop

Esta aplicación se ha realizado con la función auxiliar que se observa en la figura 86 y cuya cabecera se adjunta en forma de esquema en la siguiente figura.

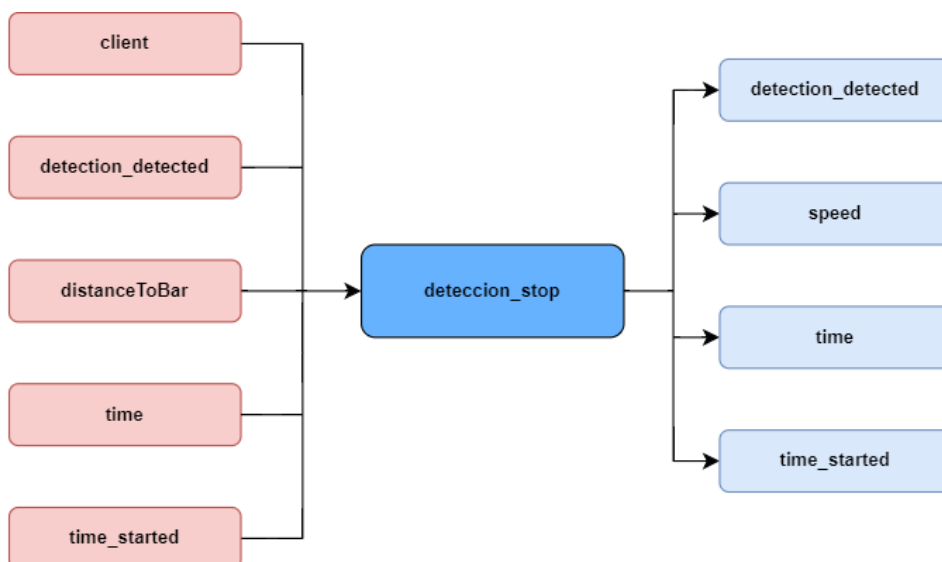


Figura 88. Esquema cabecera detección en stop

Con esta función, se busca detener el vehículo delante de la señal stop durante un tiempo determinado para posteriormente continuar con el recorrido. Para lograr esto, será necesario una

variable booleana que identifique si es la primera vez que se ve la señal de stop o no, para no quedarse atascado en la misma señal de stop, la distancia a la barra para saber dónde frenar y donde parar y el tiempo que estará detenido el vehículo.

La implementación de esta función sigue la lógica que se muestra en el siguiente esquema.

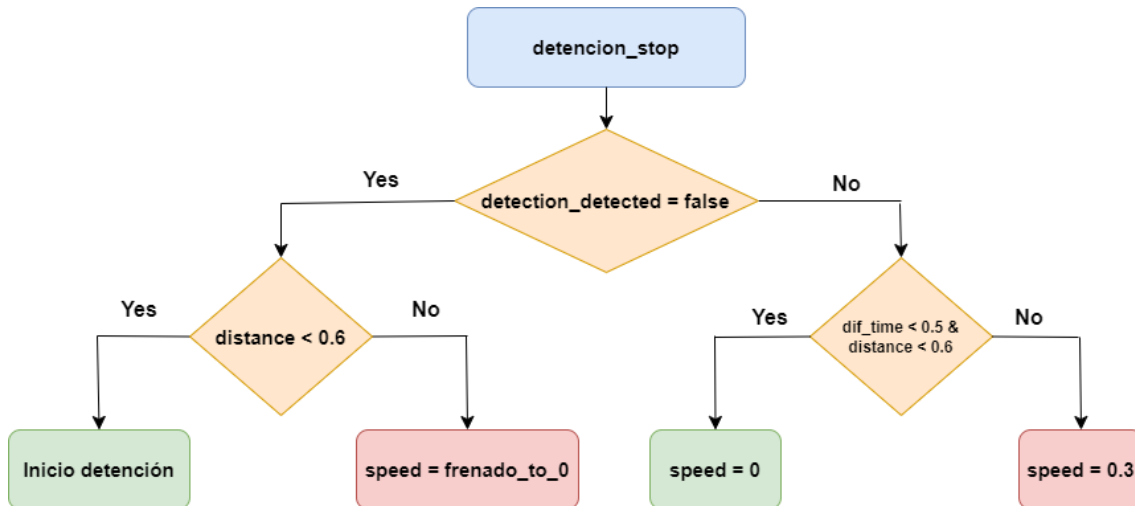


Figura. 89. Esquema lógico detencion\_stop

La implementación de este esquema lógico, con la función explicada en la figura 87, se encuentra a continuación.

```

% Primera detección del stop
if detection_detected == false
    % Distancia a la barra menor de 0.6
    if (distanceToBar < 0.6)
        % Se comienza la detección del vehículo
        time_started = sim.getSimulationTime();
        time = time_started;
        detection_detected = true;
        speed = 0;
    else
        speed = frenado_to_0(distanceToBar);
    end
else
    % Se detiene el vehículo el tiempo establecido
    if ((time - time_started < 0.5) & (distanceToBar < 0.6))
        speed = 0;
    else
        speed = 0.3;
    end
end
end
  
```

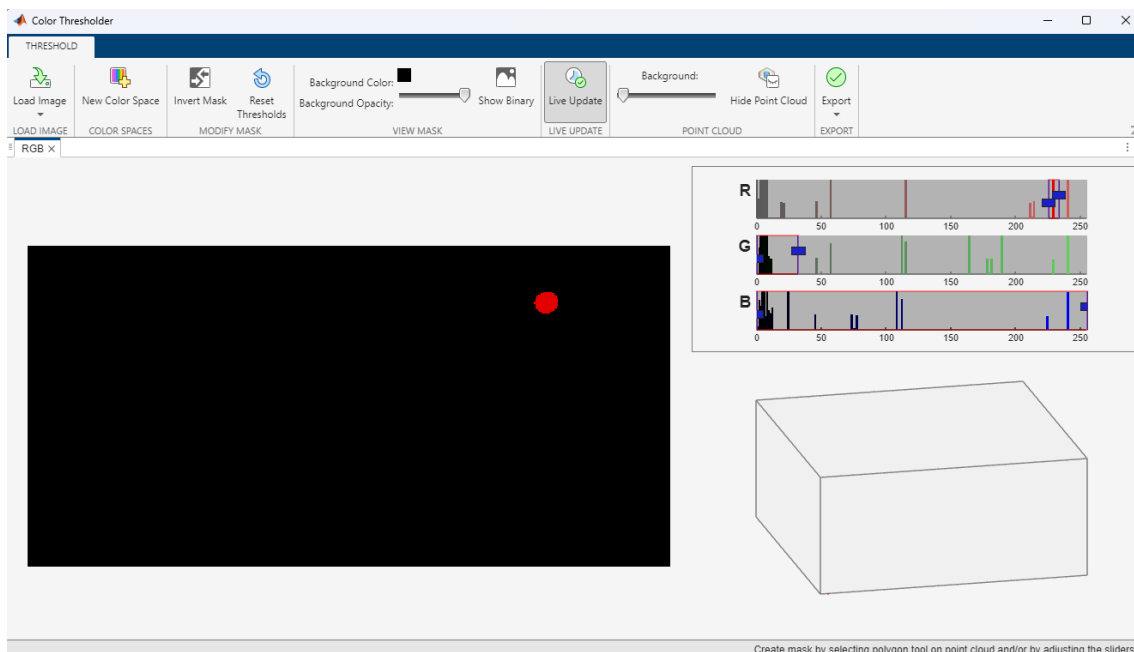
Figura 90. Implementación para la parada en el stop

Con esta función, se logrará detener el vehículo delante del stop durante el tiempo decidido y posteriormente continuar con el recorrido sin problema.

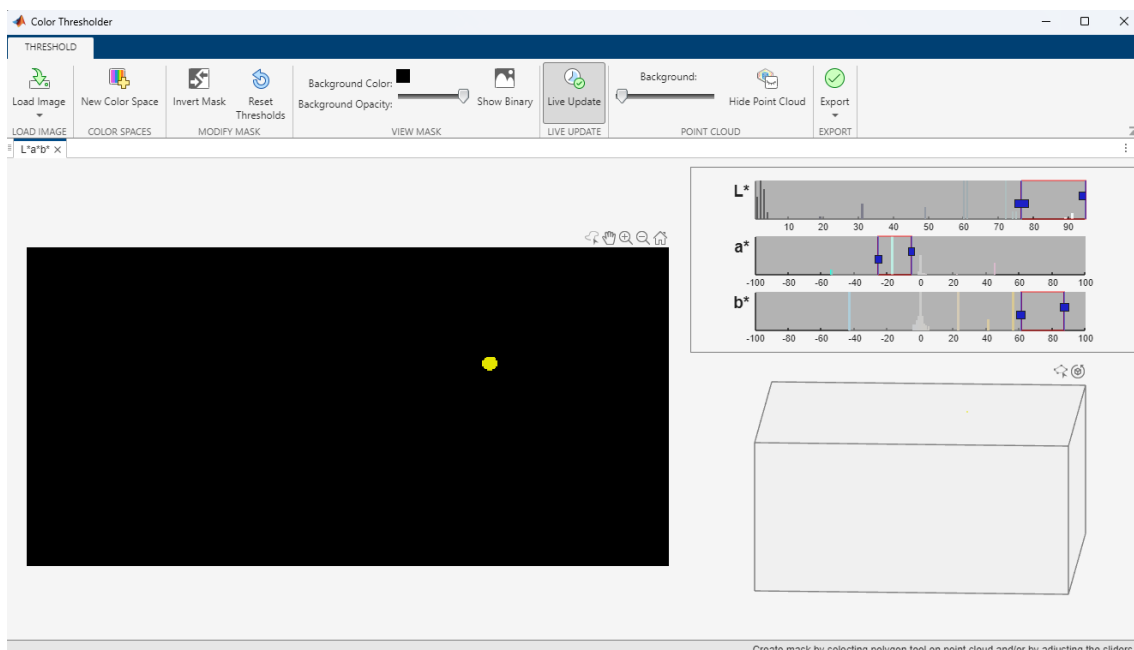
### 4.6.3.3. Semáforo

La actuación entorno a esta señal será similar a la de stop aunque en este caso habrá tres posibles casuísticas, rojo, ámbar y verde, para detectar estas, se ha utilizado la herramienta de Matlab Color Thresholder [39], donde se segmentará el color del led que esté encendido en cada caso.

A continuación, se adjuntan las 3 segmentaciones creadas y aplicadas sobre la misma imagen, para que se observen los posibles estados del semáforo.

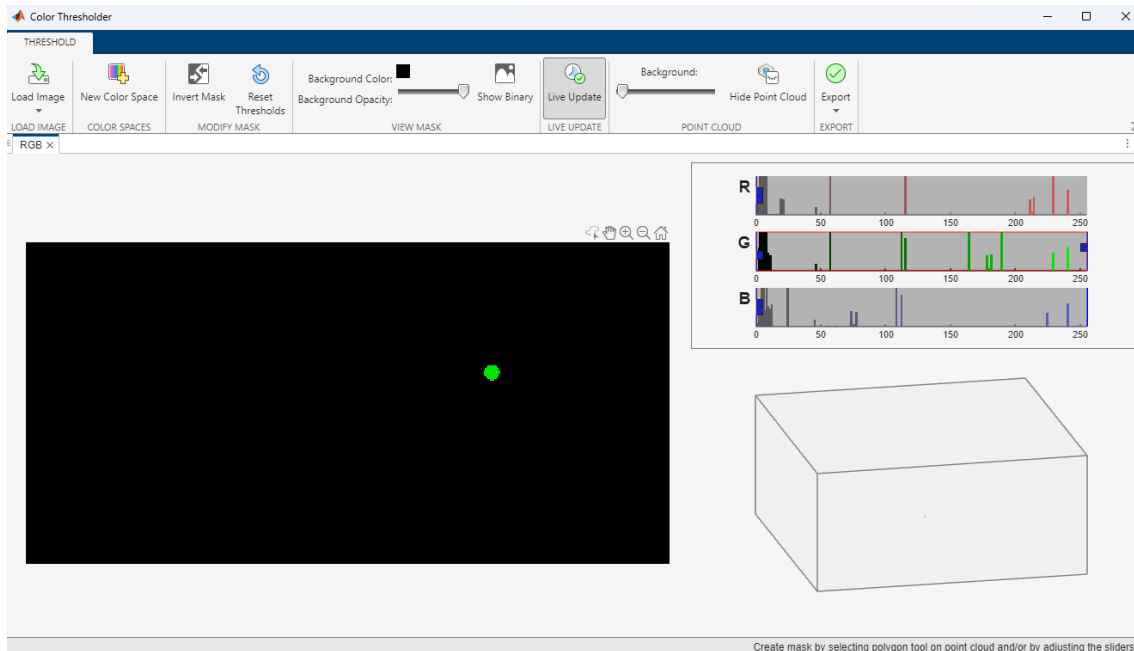


*Figura 91. Segmentación para semáforo en rojo*



*Figura 92. Segmentación para semáforo en ámbar*





**Figura 93. Segmentación para semáforo en verde**

Esta umbralización se realiza mediante la función `detector_luz` la cual implementará las segmentaciones de las figura anteriores para posteriormente medir el área resultante tras dicho recorte y por lo tanto el led que está encendido. La realización de este proceso se realiza mediante el siguiente código.

```
% Clasificación de semáforo rojo, amarillo y verde
[BW_rojo, ~] = clasi_semaforo_rojo(img);
[BW_ambar, ~] = clasi_semaforo_ambar(img);
[BW_verde, ~] = clasi_semaforo_verde(img);

% Encuentra la región con el área más grande en cada imagen
props_rojo = regionprops('table', BW_rojo, 'Area', 'BoundingBox');
props_ambar = regionprops('table', BW_ambar, 'Area', 'BoundingBox');
props_verde = regionprops('table', BW_verde, 'Area', 'BoundingBox');

% Muestra la imagen original del semáforo con el objeto más grande si el área es mayor que cero
max_area_rojo = max(props_rojo.Area);
max_area_ambar = max(props_ambar.Area);
max_area_verde = max(props_verde.Area);

if any([max_area_rojo > 0, max_area_rojo > max_area_ambar, max_area_rojo > max_area_verde])
    type = 'rojo';
elseif any([max_area_ambar > 0, max_area_ambar > max_area_rojo, max_area_ambar > max_area_verde])
    type = 'ambar';
elseif any([max_area_verde > 0, max_area_verde > max_area_rojo, max_area_verde > max_area_ambar])
    type = 'verde';
end
```

**Figura 94. Detección del color del semáforo**

Como se puede observar, esta función ha devuelto el nombre del color que ha sido detectado para posteriormente y mediante el código que se muestra en la siguiente figura tomar la acción pertinente.

```
case 'Semaforo'
    detection_detected = false;
    type = detector_luz(img);
    disp(type)
    if strcmp(type, 'rojo')
        speed = 0;
    elseif strcmp(type, 'ambar')
        speed = 0.3;
    elseif strcmp(type, 'verde')
        speed = 0.6;
    end
```

*Figura 95. Código de la actuación ante semáforos*

## 5. Detección del carril y cálculos de distancia

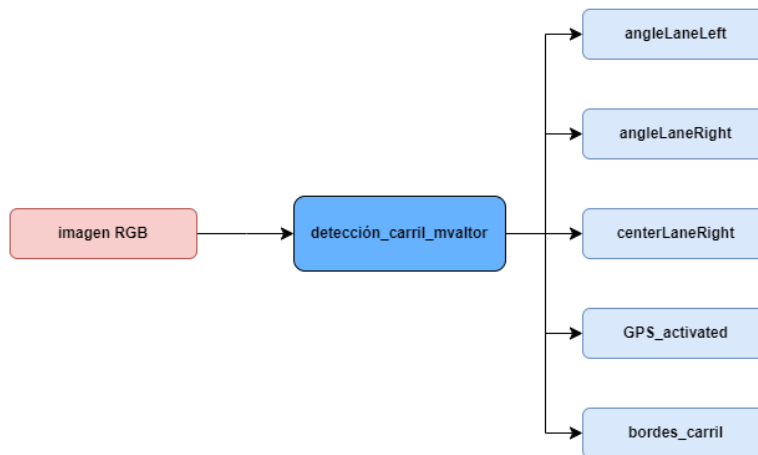
En este capítulo, se desarrollarán todos aquellos algoritmos utilizados en el procesamiento de imágenes y en todas las técnicas que constituyen este como son la morfología matemática, la umbralización por color (utilizada ya en el punto 4.6.2) o el algoritmo de Canny entre muchos otros.

Estas técnicas, se utilizarán de cara a la obtención de los ángulos de las líneas del carril de la derecha, para el posterior control del vehículo, y para la detección de la barra de las señales y semáforos, para así obtener la distancia hasta las señales.

### 5.1. Detección del carril

La detección del carril es una de las partes más importantes del algoritmo de control del vehículo, y es que, este permitirá al vehículo orientarse y circular de manera adecuada por el carril correspondiente. Es por ello, que no solo se limitará este apartado a obtener las líneas sino que se desarrollarán técnicas para analizar todo el entorno capturado y segmentarlo en partes como medida de protección extra ante posibles imperfecciones o fallos.

Para mejorar la comprensión de algoritmo de control, la función principal y todas sus funciones auxiliares se encontrarán dentro de la carpeta carril según la estructura de la figura 20. A continuación, se adjunta un esquema con las entradas y salidas que tendrá la función encargada de detectar las líneas del carril derecho.



*Figura 96. Esquema cabeceras de la función `detección_carril_mvalor`*

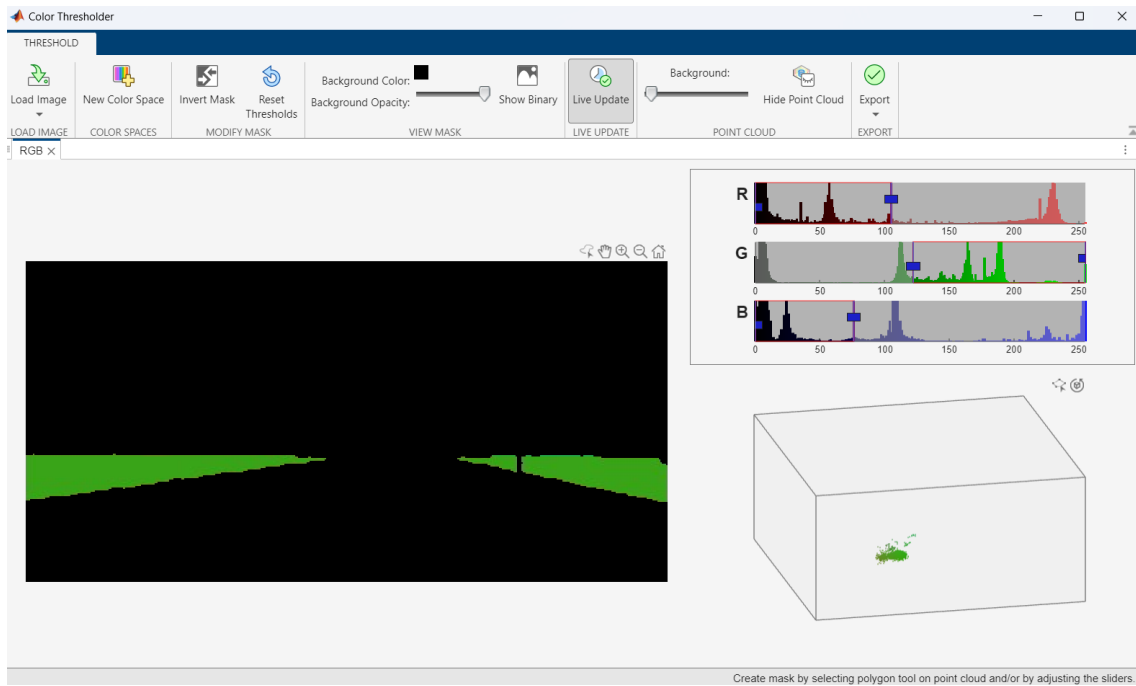
El código completo de esta función se puede encontrar en el anexo 1.9 aunque debido a la longitud y complejidad de este, en los siguiente apartados se explicarán las partes más importantes de manera más detallada y profunda.

### 5.1.1. Análisis del entorno

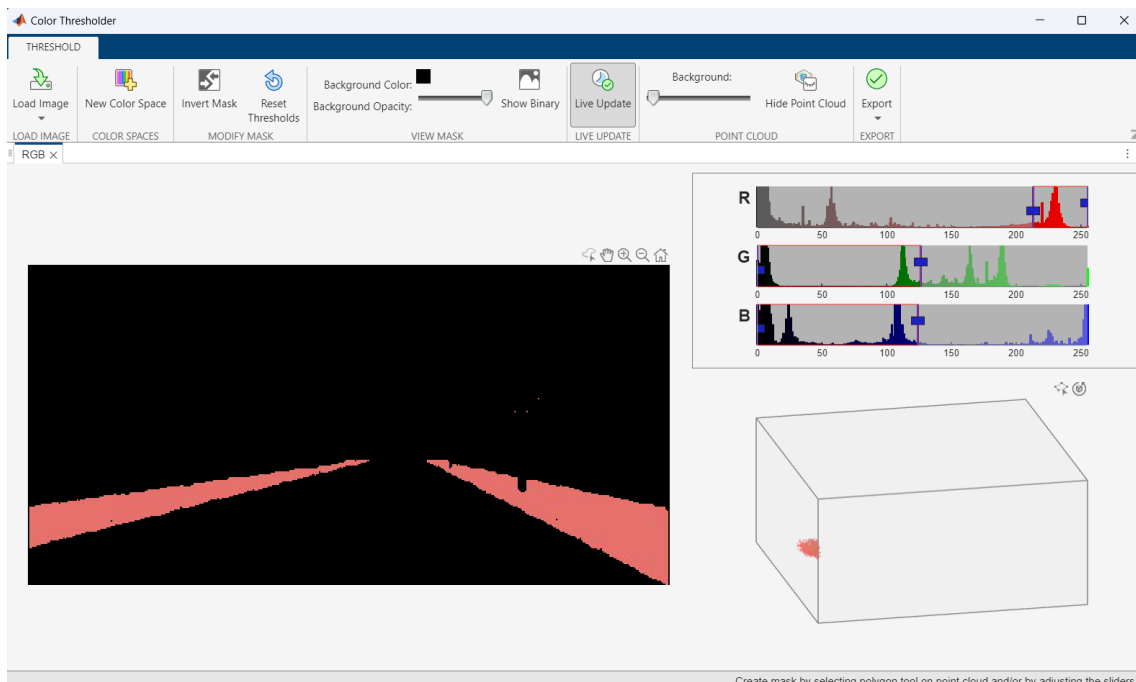
El primer paso, es la clasificación de las partes más importantes del entorno tales como el césped, el arcén, el cielo o la línea divisoria, con ello, se busca no solo aumentar la robustez del algoritmo sino preparar el modelo para futuras ampliaciones. Al analizar toda la imagen, y no solo el carril en sí, se logra tener una visión más global de la situación y no limitar el campo de interés al entorno más cercano del vehículo, lo que será decisivo en implementaciones de seguridad y prevención de riesgos.

Esta primera etapa de clasificación se ha realizado mediante la herramienta Color Thresholder [36], de manera individual para cada parte del escenario y combinándose todas ellas en esta función hasta obtener finalmente con el carril. Cabe destacar, que en la clasificación de la línea divisoria se ha utilizado morfología matemática para aumentar la robustez de esta umbralización.

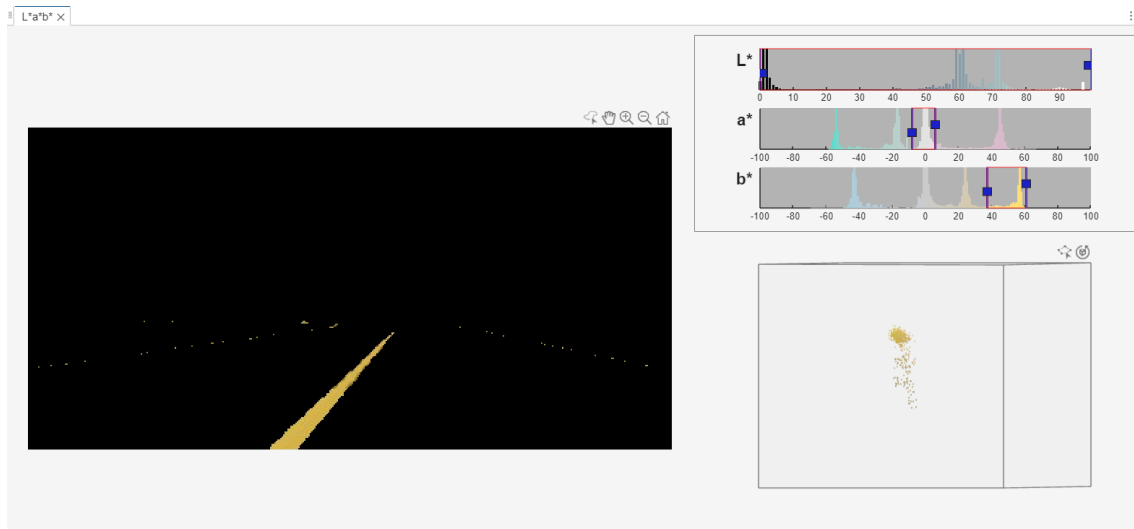
A continuación, se adjuntan las segmentaciones individuales de las distintas partes del escenario desde la interfaz de la aplicación del Color Thresholder utilizada.



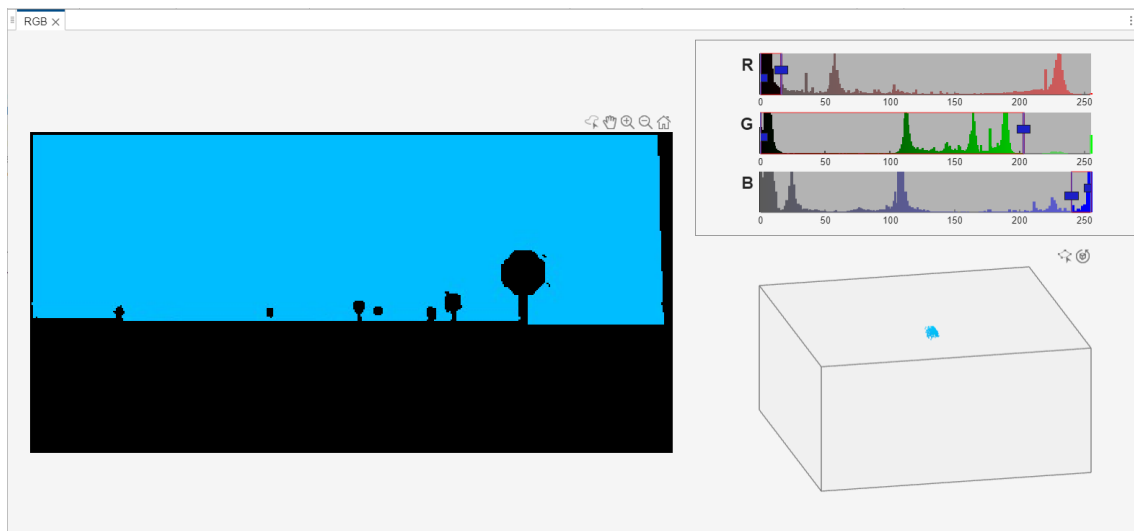
*Figura 97. Segmentación por color del césped*



*Figura 98. Segmentación por color del arcén*



**Figura 99. Segmentación por color de la línea divisoria**



**Figura 100. Segmentación por color del cielo**

La totalidad de las funciones generadas a través de esta herramienta, se almacenan como funciones individuales y se combinarán para obtener la imagen sobre la que se empezará a trabajar una vez esta ha sido binarizada, siendo el código utilizado y las imágenes resultantes las siguientes.

```
% Aplicar las funciones de clasificación
img_arcen = clasi_arcen(IMG);
img_cielo = clasi_cielo(IMG);
img_cesped = clasi_cesped(IMG);
img_linea_divisoria = clasi_linea_divisoria(IMG);
disk = strel("square",3);
img_linea_divisoria = imdilate(img_linea_divisoria,disk);

% Combinar las imágenes binarias usando la operación OR
img_final = img_arcen | img_cielo | img_cesped | img_linea_divisoria;

% Crear subplot de 2x3
figure;
subplot(2, 3, 1);
imshow(~img_final);
title('Imagen Final');

subplot(2, 3, 2);
imshow(img_arcen);
title('Arcén Binario');

subplot(2, 3, 3);
imshow(img_cielo);
title('Cielo Binario');

subplot(2, 3, 4);
imshow(img_cesped);
title('Césped Binario');

subplot(2, 3, 5);
imshow(img_linea_divisoria);
title('Línea divisoria binaria');
```

Figura 101. Código para la clasificación del entorno

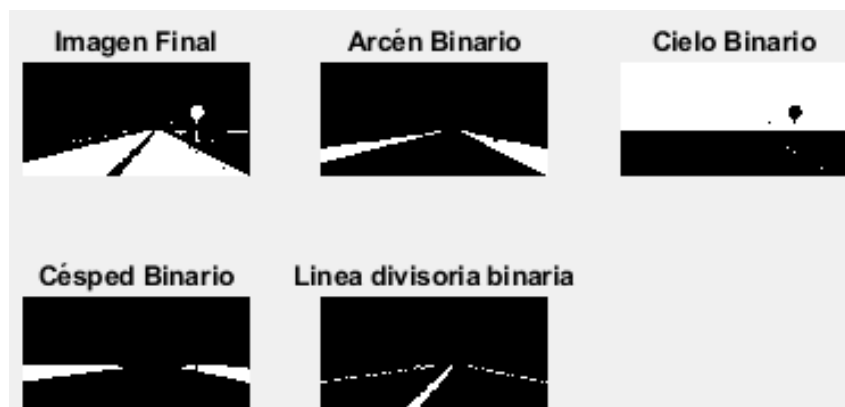


Figura 102. Clasificación final y conjunta

## 5.1.2. Eliminación del ruido

Una vez se ha obtenido ya la imagen resultante de la umbralización mencionada en el apartado anterior, se procede a trabajar con esta, la primera parte, es la eliminación de la parte superior de los bordes laterales como se adjunta a continuación.

```
% Obtiene el tamaño de la imagen
[h, w] = size(img_final);

% Define el número de píxeles a eliminar de los bordes
cut = 20;

% Pone a 0 los píxeles de los lados excepto los 40 últimos de abajo que
% forman parte de la carretera
img_final(1:cut, :) = 0; % Arriba
img_final(1:(h-110), 1:cut) = 0; % Izquierda
img_final(1:(h-110), w-cut+1:w) = 0; % Derecha
```

*Figura 103. Código para la eliminación de los bordes laterales*

Esta etapa se incluye, con el fin de eliminar posible ruido de la imagen y aunque no será estrictamente necesaria, otorga al algoritmo un mayor grado de robustez. Más en particular, este ruido podrá deberse a edificios o similares los cuales son indeseados para esta aplicación y por lo tanto conviene eliminarlos lo antes posible centrando así el procesado en la parte más frontal del vehículo.

Posteriormente, se hará uso de una función auxiliar encargada de eliminar todas aquellas partes que no pertenecen al carril y que no han sido eliminadas por el recorte anterior. Para ello, se usa el comando `regionprops` [51], que devolverá el tamaño de los objetos detectados, lo que permite diferenciar rápidamente entre los dos objetos principales, es decir, los dos carriles, y el ruido de la imagen.

Como no siempre habrá dos carriles en la imagen, y como medida extra de robustez, solo se seleccionará un segundo objeto si este es al menos un 40% del primero lo que asegura que este segundo objeto no pueda ser debido al ruido, el código de la función utilizada se muestra a continuación.

```
function img_final = Two_Biggest_Objects(img_final)

% Calcular las propiedades de las regiones en la imagen final
stats = regionprops(img_final, 'Area', 'PixelIdxList');

% Obtener todas las áreas de los objetos
areas = [stats.Area];

% Encontrar el índice del objeto con el área más grande
[max_area, max_idx] = max(areas);

% Calcular el área mínima requerida para guardar el segundo objeto
area_minima = round(0.4 * max_area);

% Crear una imagen vacía del mismo tamaño que img_final
img_final = zeros(size(img_final));

% Obtener la máscara del objeto con el área más grande
img_final(stats(max_idx).PixelIdxList) = 1;

% Recorrer todos los objetos
for i = 1:numel(stats)
    % Verificar si el área del objeto actual es mayor que area_minima
    if stats(i).Area >= area_minima
        % Si es así, guardar el objeto en la imagen final
        img_final(stats(i).PixelIdxList) = 1;
    else
        % Si no, borrar el objeto de la imagen final
        img_final(stats(i).PixelIdxList) = 0;
    end
end
end
```

*Figura 104. Función para obtener los dos objetos principales*

En la siguiente figura, se adjunta una comparativa de la imagen antes y después de aplicar esta función.



*Figura 105. Imagen con los objetos principales*

Aunque la mayor parte del ruido ha sido eliminado con éxito, será importante hacer uso del comando `imfill` para rellenar los pequeños huecos que aparecen dentro de los carriles, siendo la imagen resultante la siguiente.





*Figura 106. Aplicación del comando imfill*

Como se puede ver, la imagen ya está completamente segmentada de tal forma que solo se tienen los dos carriles, por lo que el siguiente paso será ya centrarse en el carril de circulación, en este caso el derecho.

### **5.1.3. Trabajo sobre la línea divisoria**

En este apartado, se centrará el trabajo en la línea divisoria, y es que, además de ser uno de los dos ángulos que se quieren detectar, servirá para eliminar el carril de la izquierda ya que ese no es de interés para la circulación. Esta eliminación del carril izquierdo se realiza mediante la detección del punto más alto de la línea divisoria para posteriormente suprimir los píxeles de su izquierda.

Cabe destacar, que otro mecanismo de robustez que se ha añadido en esta sección, es el caso en el que no hay línea divisoria, en este, se interpretará que solo existe un carril y se circulará sin ningún problema. A continuación, se adjunta el código con los bucles anidados mediante los cuales se obtienen las coordenadas del píxel más alto de la línea divisoria.

```
% Eliminar imperfecciones la línea divisoria
stats_linea = regionprops(img_linea_divisoria, 'Area', 'PixelIdxList');

% Obtener todas las áreas de los objetos
areas_linea = [stats_linea.Area];
if ~isempty(stats_linea)
    % Obtener todas las áreas de los objetos
    areas_linea = [stats_linea.Area];

    % Encontrar el índice del objeto con el área más grande
    [max_area_linea, max_idx_linea] = max(areas_linea);

    % Crear una imagen vacía del mismo tamaño que img_linea_divisoria
    img_linea_divisoria = zeros(size(img_linea_divisoria));

    % Obtener la máscara del objeto con el área más grande
    img_linea_divisoria(stats_linea(max_idx_linea).PixelIdxList) = 1;

% Cálculo del límite de la línea divisoria
h_lim = 0;
w_lim = 0;
count = 0;
[h, w] = size(img_linea_divisoria);
% Encontrar las coordenadas del píxel más alto con valor 1
for i=h:-1:1
    for j=w:-1:1
        % Si se detecta línea almacenar la altura de ese píxel
        if(1 == img_linea_divisoria(i,j))
            count = count + 1;
            if(count >= 5)
                count = 0;
                h_lim = i;
                w_lim = j;
            end
        end
    end
end
else
    % En caso de no encontrar ninguna línea, asignar valores predeterminados
    h_lim = 0;
    w_lim = 0;
end
```

**Figura 107. Código para obtener las coordenadas del punto superior de la línea divisoria**

Una vez estas coordenadas han sido obtenidas, se realizará el mismo procedimiento de bucles anidados solo que en este caso para eliminar los píxeles de la izquierda mediante el código de la siguiente figura

```
% Eliminar el carril de la izquierda
for i=1:h
    for j=w:-1:1
        % Si se detecta línea
        if(1 == img_linea_divisoria(i,j))
            while(j>1)
                img_final(i,j) = 0;
                j = j - 1;
            end
        end
    end
end
```

**Figura 108. Recorte del carril izquierdo**

La imagen una vez se ha suprimido el carril izquierdo es la siguiente.



*Figura 109. Imagen resultante tras recortar el carril izquierdo*

Una vez se ha finalizado esta etapa ya se podrá pasar a la obtención de los ángulos del carril ya que el resto de la imagen ha sido eliminado.

### 5.1.4. Obtención de los bordes laterales

En esta etapa, se volverá a utilizar morfología matemática para eliminar aquellos objetos con un área menor de 50 píxeles y es que cada vez que suprimimos una parte de la imagen, en este caso en particular el carril izquierdo, es importante asegurarse de no dejar restos en el proceso. El comando utilizado será el `bwareaopen` [52], este comando elimina aquellos objetos menores del área especificada sin alterar el resto de la imagen, lo que es ideal para esta implementación.

Posteriormente, para la obtención de los bordes laterales del carril, se hará uso del comando de Matlab `Edge` [53] y más en particular del algoritmo de Canny [54] mediante el cual se obtienen las líneas laterales del objeto.

A continuación, se adjuntan las líneas de código que realizan lo explicado con anterioridad junto con la imagen resultante de aplicar estas.

```
% Detección de los bordes del carril  
img_final = bwareaopen(img_final,50);  
bordes_carril = edge(img_final,'Canny');
```

*Figura 110. Implementación del algoritmo de Canny*



*Figura 111. Implementación del algoritmo de Canny*

Como cabría esperar, además de las líneas laterales también se han obtenido las líneas horizontales del borde superior, sin embargo, estas no serán relevantes para el control del vehículo y por lo tanto convendrá eliminarlas en las sucesivas etapas.

### 5.1.5. Adaptación para curvas

Una de las partes más conflictivas de este tipo de algoritmos, es su comportamiento en tramos de curvas, sin embargo, una línea curva está compuesta por infinitos tramos rectos. Teniendo esto en cuenta, se eliminará la parte superior de la imagen hasta contar únicamente con los 15 píxeles inferiores de la imagen.

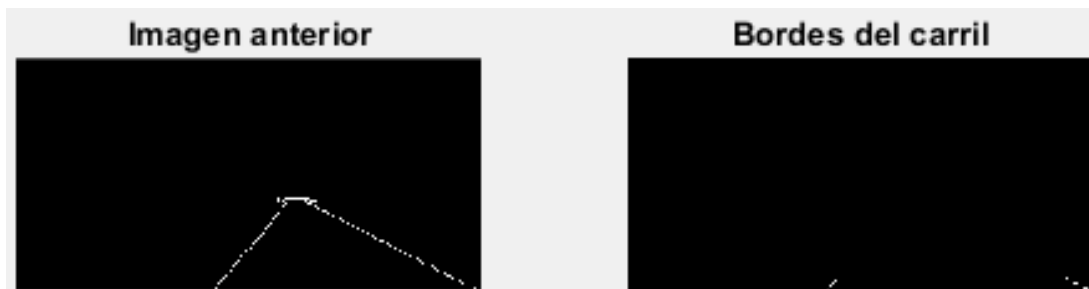
Este valor dependerá de la velocidad y el tiempo de reacción del algoritmo ya que cuanto menor sea este recorte, más tiempo tendrá el vehículo para reaccionar. En este proyecto en concreto, se ha decidido un valor de 15 píxeles aunque este valor podrá variar dependiendo de las circunstancias en las que circule el vehículo.

De esta forma, se logra superar este problema ya que al tener en cuenta solo los tramos más cercanos al vehículo, este será capaz de corregir su orientación rápidamente para girar el vehículo en caso de ser necesario.

El código utilizado para este recorte será similar al utilizado para el recorte de los bordes laterales y la imagen resultante se adjuntan a continuación.

```
% Recortar de la parte superior de la imagen  
bordes_carril(1:(h-15), :) = 0;
```

*Figura 112. Código para la adaptación a curvas*



*Figura 113. Bordes laterales una vez aplicada la adaptación a las curvas*

### 5.1.6. Obtención de los ángulos y del centroide derecho

Una vez se ha llegado hasta este punto, ya se podrán obtener los ángulos de las líneas del carril por el que el vehículo ya está circulando. En esta etapa, es donde más medidas protectoras se han aplicado, ya que al ser la última parte del algoritmo cualquier fallo que se haya podido escapar en las etapas anteriores debe ser eliminado aquí antes de poder alterar el comportamiento del vehículo.

Lo primero que se hará para obtener los ángulos que se requieren, es clasificar y ordenar los objetos de la imagen binaria, este procedimiento se realiza con las funciones `bwlabel`, para el

etiquetado, y regionprops, para la obtención de sus características, a continuación, se adjunta la implementación de estos comandos.

```
% Obtenemos los ángulos de los bordes del carril
[L, ~] = bwlabel(bordes_carril);
props = regionprops(L, 'Centroid', 'MajorAxisLength', 'Orientation');
[~, index] = sort([props.MajorAxisLength], 'descend');
filteredOrientations = [props(index).Orientation];
centerLaneRight = [0, 0];
```

*Figura 114. Identificación y clasificación de las líneas*

Una vez se han obtenido las orientaciones de los laterales del carril, se presentan tres posibles casos:

- **No se han detectado suficientes líneas:** En este caso, se estará en una intersección o en una rotonda por lo que se activará el control del GPS hasta que este cambio de trayectoria se complete.
- **Se han detectado dos líneas:** Caso ideal y más común, el ángulo izquierdo será el positivo y el derecho el negativo, por el sentido de la orientación.
- **Se han detectado más de dos líneas:** Ha habido un error en la umbralización que debe ser resuelto, para ello, el ángulo izquierdo será el más cercano a  $35^\circ$ , este valor se debe a la orientación de la cámara y el punto de fuga del carril, y el derecho será el más cercano a  $-35^\circ$ . Estos ángulos se han seleccionado debido a la perspectiva de la cámara aunque si esta cambiase deberían ser recalculados de la manera más precisa posible.

Para tratar todos estos casos, se ha implementado un if que contemple todas estas posibilidades. Además, el procedimiento para cuando se detecten más de dos líneas es compatible con el caso ideal donde tan solo se tendrán dos orientaciones por lo que se combinarán dichos escenarios para lograr así un proceso computacional más eficiente.

Finalmente, para calcular el centroide de la línea derecha del carril se volverá a usar el comando regionprops, a continuación, se adjuntan los fragmentos de código para la obtención de los ángulos del carril y del centroide respectivamente.

```
if numel(props) > 1
    % Robustez ante más de un ángulo por si hubiese errores y se detectase más
    % de un borde debido a un fallo en el edge

    for i = 1:numel(filteredOrientations)

        % Filtrar y seleccionar el ángulo más cercano a -35 para el carril derecho
        if filteredOrientations(i) >= -85 && filteredOrientations(i) <= -5
            if abs(filteredOrientations(i) + 35) < abs(dif_right)
                angleLaneRight = filteredOrientations(i);
                dif_right = abs(filteredOrientations(i) + 35);
            end
        end

        % Filtrar y seleccionar el ángulo más cercano a 35 para el carril izquierdo
        if filteredOrientations(i) >= 5 && filteredOrientations(i) <= 85
            if abs(filteredOrientations(i) - 35) < abs(dif_left)
                angleLaneLeft = filteredOrientations(i);
                dif_left = abs(filteredOrientations(i) - 35);
            end
        end
    end

    GPS_activated = false;
else
    % Se activa el GPS por haberse detectado un cruce o rotonda
    GPS_activated = true;
end
```

Figura 115. Código para la identificación de los ángulos del carril

```
% Calcular el centroide del carril derecho
stats = regionprops(L, 'Centroid', 'Orientation');
for j = 1:numel(stats)
    if stats(j).Orientation == angleLaneRight
        centerLaneRight = stats(j).Centroid;
        break; % Detener la iteración después de encontrar el centroide del carril derecho
    end
end
```

Figura 116. Obtención del centroide del carril derecho

Como se puede observar en las imágenes anteriores, todo el código está preparado para ser funcional en caso de que se detecten más de una línea lo que es fundamental para crear un algoritmo robusto en todo tipo de casos.

### 5.1.7. Control del vehículo

Una vez se han obtenido los ángulos de las líneas laterales del carril, se procede con el control del vehículo en función de esta información, para ello, se utilizarán dos funciones auxiliares. Cabe mencionar, que este algoritmo de control se ha basado en el algoritmo utilizado en uno de los trabajos mencionados en el estado del arte [20], ya que es la forma más eficiente de controlar un vehículo de tipo Ackerman.

Para realizar este control Ackerman, se hará uso de dos funciones distintas, la primera de ellas para la obtención de los ángulos de giro, y la segunda para transmitir esta información al vehículo durante la simulación.

### 5.1.7.1. Obtención de los ángulos de giro

El control Ackerman, es el modelo seguido por la inmensa mayoría de los coches en la actualidad, este, se basa en el principio por el cual las ruedas exteriores deben girar más que las interiores al tener estas una trayectoria mayor. De esta forma, se logra reducir el deslizamiento y mejorar la estabilidad del vehículo y el rendimiento de las ruedas lo que hace de este sistema una pieza fundamental de la industria automotriz.

En la siguiente figura se muestra de manera gráfica este principio de funcionamiento.

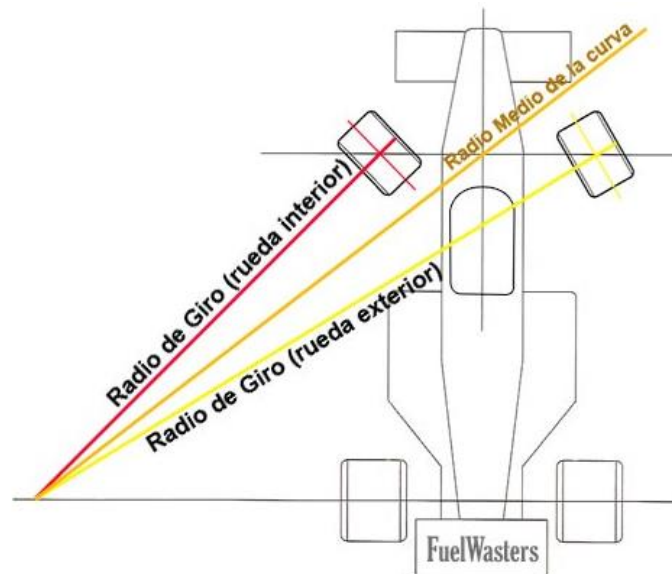
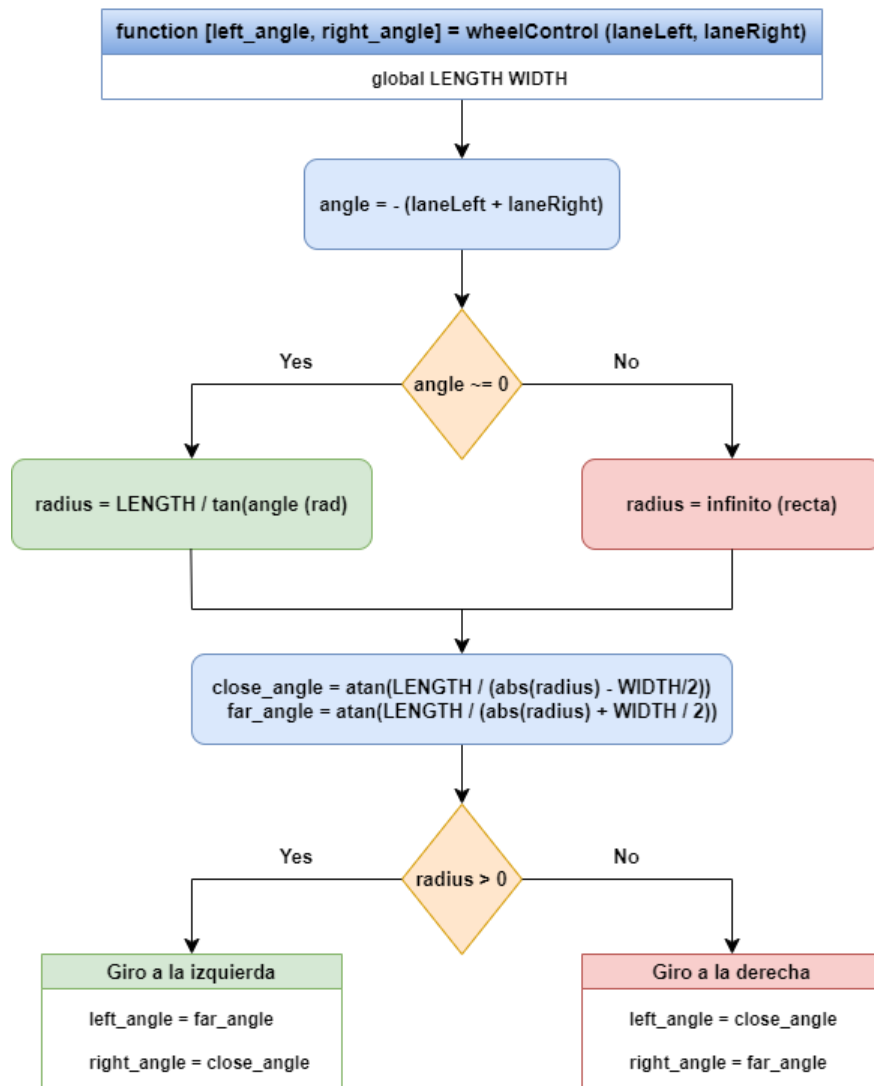


Figura 117. Esquema funcionamiento control Ackerman [55]

En este proyecto, se usará una función auxiliar para la conversión de los ángulos de las líneas del carril a ángulos de giro para cada una de las ruedas, el esquema de esta función se adjunta a continuación.



*Figura 118. Esquema función para los ángulos de giro*

Como se puede observar en el esquema, esta función calculará la diferencia entre las orientaciones de las dos líneas para posteriormente, seleccionar el sentido de giro del vehículo y los ángulos de giro de cada rueda. Cabe destacar, que en el caso de estar en una recta, ambos ángulos de giro serán 0 debido a la arquitectura planteada.

La implementación de este esquema en el código se realiza mediante la siguiente función.



```
function [left_angle, right_angle] = wheelControl(laneLeft, laneRight)
    LENGTH = 0.316;
    WIDTH = 0.213;

    % Estandar (recta)
    angle = -(laneLeft + laneRight);
    if angle ~= 0
        radius = LENGTH / tan(deg2rad(angle));
    else
        radius = Inf;
    end
    close_angle = atan(LENGTH / (abs(radius) - WIDTH/2));
    far_angle = atan(LENGTH / (abs(radius) + WIDTH / 2));

    if radius > 0
        % Giro hacia la izquierda
        left_angle = far_angle;
        right_angle = close_angle;
    else
        % Giro hacia la derecha
        left_angle = -close_angle;
        right_angle = -far_angle;
    end
end
```

Figura 119. Implementación para los ángulos de giro

### 5.1.7.2. Transmisión de la información

Una vez los ángulos de giro han sido obtenidos, se hará uso de otra función mediante la cual se transmite al vehículo la información en tiempo real durante la ejecución. Esta función, servirá simplemente como intermediaria y se utilizará para transmitir la información de manera sencilla y eficaz.

```
function setCarControl(sim, steer, lights, speed, leftAngle, rightAngle, lightOption)
    steer{2}{1} = leftAngle;
    steer{2}{2} = rightAngle;
    steer{1} = speed;
    lights{1} = lightOption;
    data = {steer, lights};
    sim.callScriptFunction('set_state@/Car', sim.scripttype_childscript, data);
end
```

Figura 120. Implementación para transmitir la información

### 5.1.8. Pruebas de estrés

Una vez se han implementado todas las funciones, y con el fin de comprobar si el algoritmo es verdaderamente robusto, se ha sometido al algoritmo de control a dos pruebas distintas donde se pone a prueba este antes escenarios extremos. Para realizar estas pruebas, se ha modificado ligeramente el algoritmo de segmentación del carril y más en particular la función auxiliar `detección_carril_mvalor`, apartado 5.1, para que en caso de detectar solo una línea considere el otro valor como 0. De esta forma, se consigue probar el algoritmo en situaciones tan dificultosas que ni siquiera serán necesarias en la implementación final por lo que si estas son superadas se podrá afirmar que el algoritmo es muy eficaz y robusto.

A continuación, se adjunta la parte de la función modificada para el propósito explicado anteriormente.

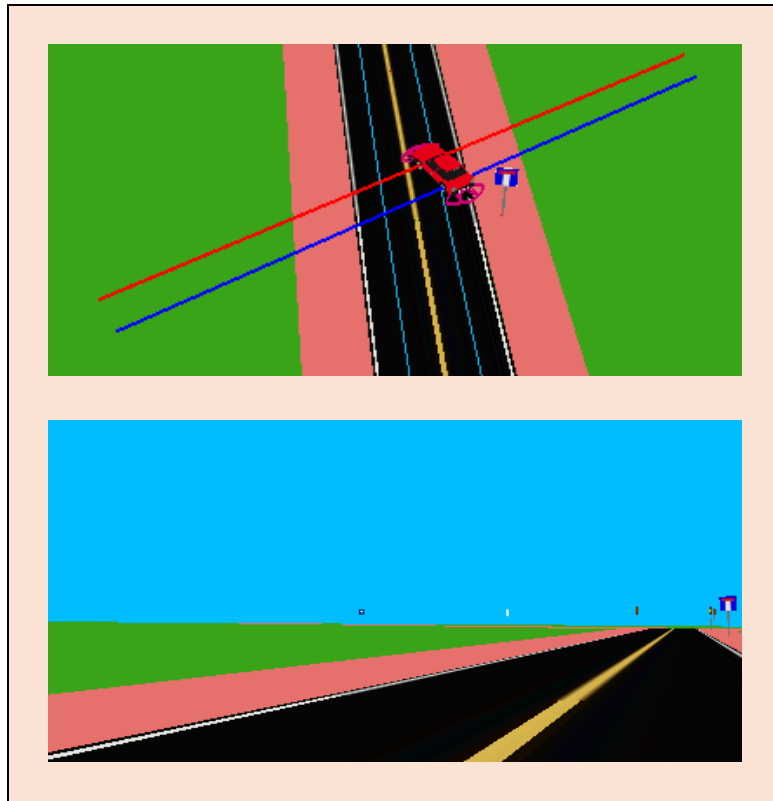
```
if ~isempty(props)
    % Robustez ante más de un ángulo por si hubiese errores y se detectase más
    % de un borde debido a un fallo en el edge
    for i = 1:numel(filteredOrientations)
```

*Figura 121. Modificación para las pruebas del carril*

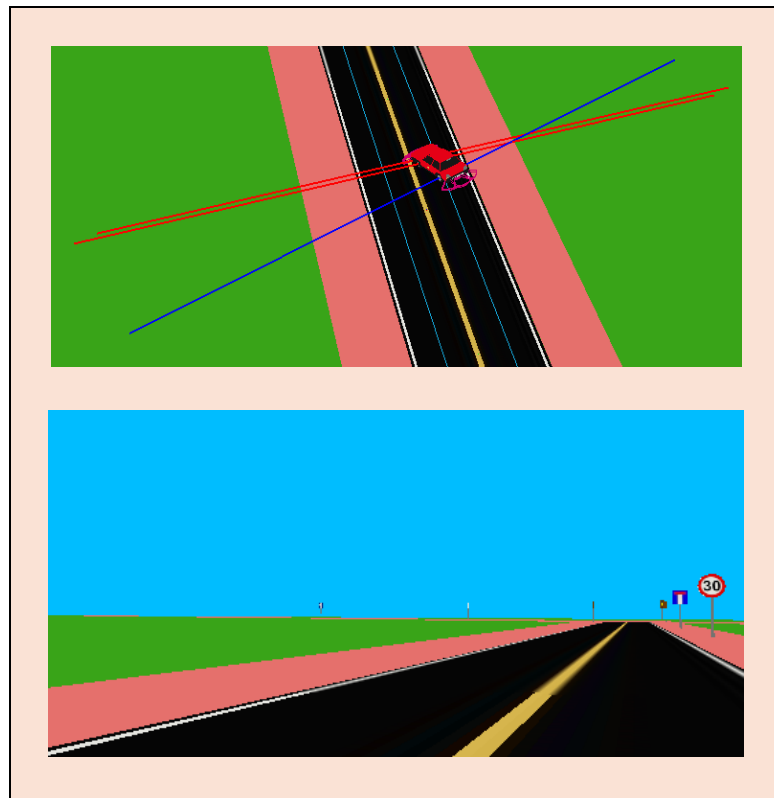
El script para el testeo de este algoritmo se encuentra en el anexo 1.10 y en los siguientes subapartados, se desarrollará en profundidad cada uno de estos test con su debida explicación y los resultados obtenidos en estos.

### 5.1.8.1. Entrada en el carril

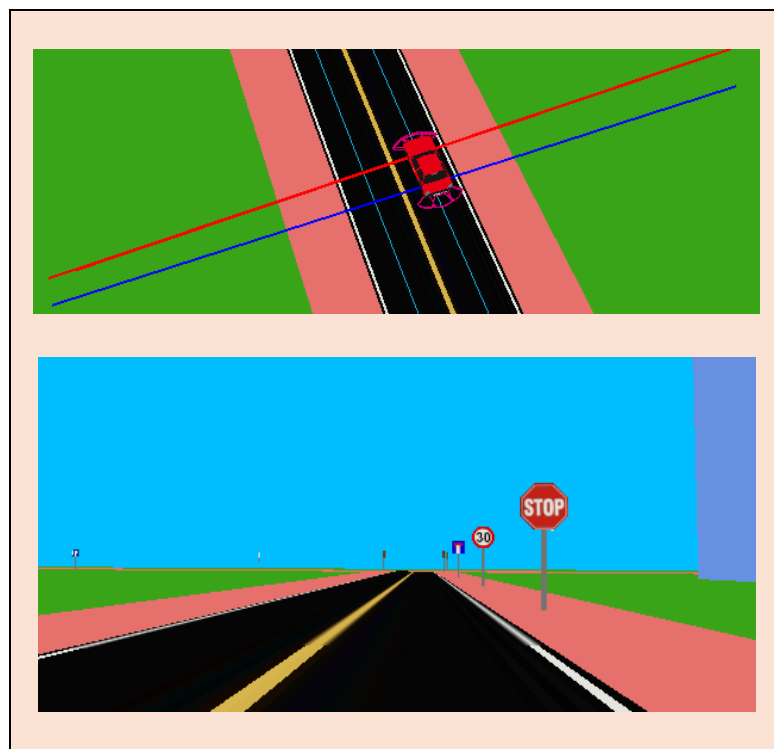
En el primer test, se colocará el vehículo desviado respecto el carril y este deberá girar lo suficiente para poder continuar con la trayectoria prevista y corregir así esta desviación, a continuación, se adjuntan varias capturas obtenidas durante la realización de esta test.



*Figura 122. Inicio del test*



*Figura 123. Reconducción de la situación*



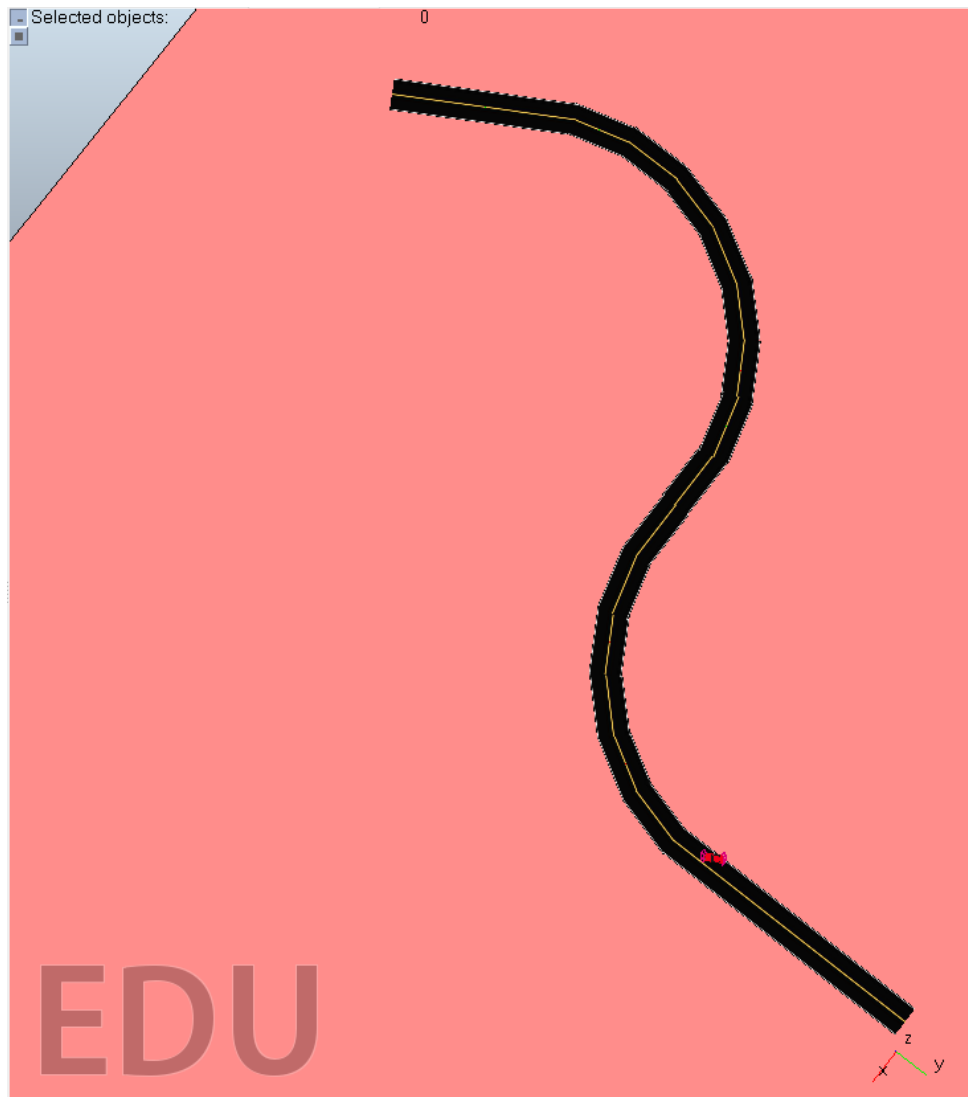
*Figura 124. Dirección centrada y restablecimiento del seguimiento de la trayectoria*

Como se ha podido observar en las figuras anteriores, el test ha sido superado en una situación donde el vehículo cuenta con una desviación de 45 grados respecto de la carretera sin ayudas externas por lo que el resultado de este test será positivo.

### 5.1.8.2. Funcionamiento en curvas

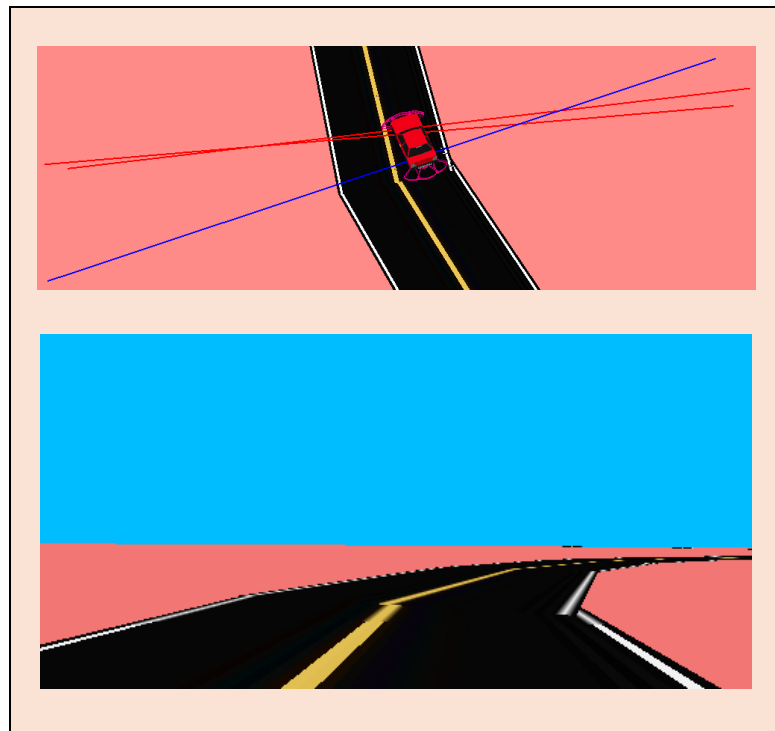
Como ya se mencionó en apartados anteriores, la mayor dificultad que presenta este tipo de algoritmos es su funcionamiento en tramos de curvas ya que puede presentar fallos al no encontrar una orientación concreta.

Por ello, se ha creado un escenario auxiliar con dos tramos de curvas, una en cada sentido, para comprobar si el algoritmo diseñado es funcional en este tipo de escenarios. A continuación, se adjunta el plano cenital de la escena creada para realizar esta prueba de estrés.

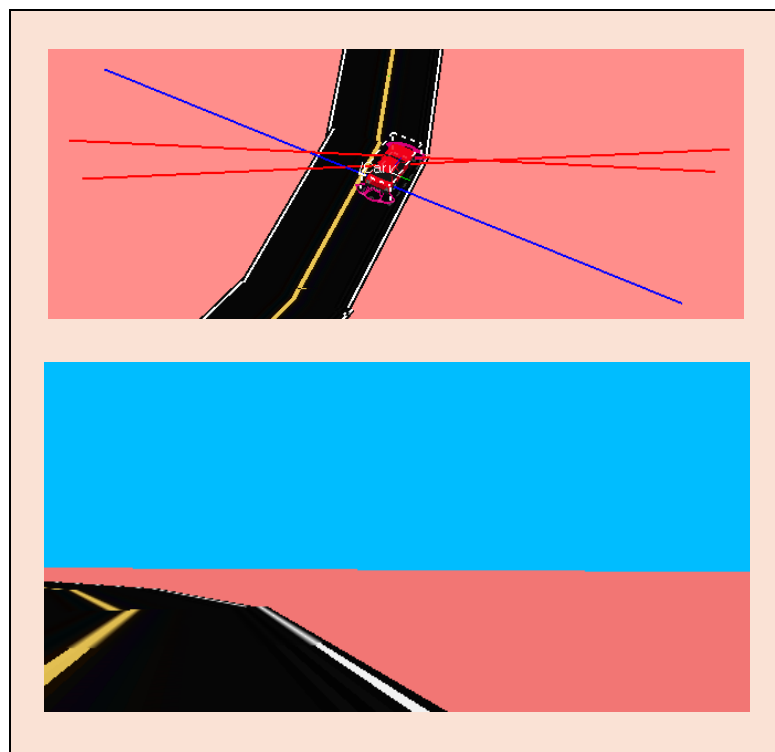


*Figura 125. Escena para realizar el test en curvas*

En las siguientes figuras se muestran distintas instantáneas tomadas durante la ejecución de este test donde se puede observar al vehículo tomando las curvas.



*Figura 126. Test de detección del carril en una curva de derechas*



*Figura 127. Test de detección del carril en una curva de izquierdas*

Como se puede observar, el vehículo es capaz de girar en ambos sentidos con precisión y sin fallos durante el proceso, quedando demostrada así la robustez de este algoritmo ante todo tipo de situaciones.

## 5.2. Detección de la barra de la señalización

La otra parte del algoritmo donde se utilizan estas técnicas de segmentación por color, o morfología matemática que ya han sido mencionadas y explicadas a lo largo del punto 5.1, será la detección de la barra de las señales para el posterior cálculo de la distancia.

Esta distancia puede ser obtenida de distintas formas tales como, otro tipo de sensores o utilizando la detección de las señales y utilizando las dimensiones reales de estas en la legislación local. Sin embargo, se ha optado por una opción alternativa, la detención de la barra de soporte de las señales, ya que no cerrará la puerta a una posible expansión a territorios con otras legislaciones y además ofrece la posibilidad de seguir enfocando este proyecto entorno a los algoritmos de visión.

Teniendo en cuenta lo anteriormente mencionado, se utilizará una función auxiliar, la cual analizará la imagen y devolverá el valor a la barra más cercana, en caso de haber sido detectada. Antes de entrar más en profundidad en la función y su funcionamiento, es importante conocer las entradas y salidas de esta de cara a conocer su objetivo, un esquema con esta información se encuentra a continuación.



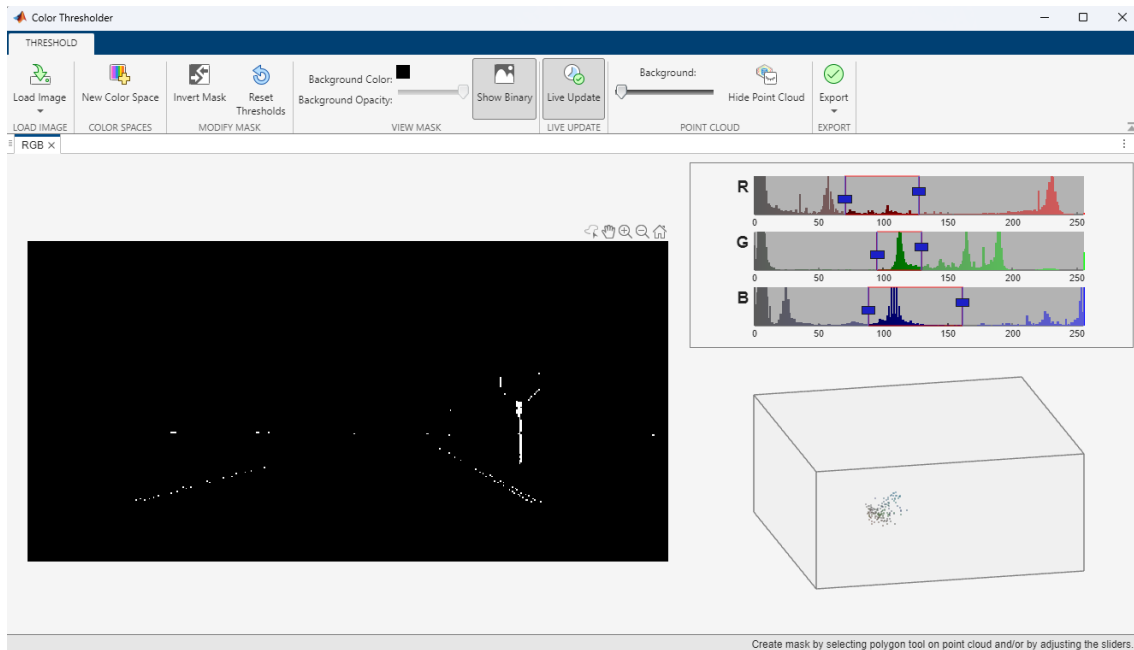
*Figura 128. Esquema de la cabecera de la función detectBar*

En esta función, al igual que en los casos que ya han sido mencionados con anterioridad, se han añadido mecanismos de robustez a lo largo de la detección que se irán comentando a lo largo de la explicación.

En la primera parte de esta función, y aunque no debería ser necesario por la arquitectura elegida en la detección de las señales, se recortará el carril izquierdo de manera análoga a lo explicado en el punto 5.1.3 y más en particular a las figura 107 y 108.

Posteriormente, se volverá a utilizar la herramienta Color Thresholder [36] para en este caso, segmentar lo máximo posible la imagen y que solo se devuelva la barra. En la realidad, esto es enormemente impreciso por lo que habrá que utilizar técnicas de depuración para aumentar la calidad de esta umbralización.

A continuación, se adjunta la interfaz de esta herramienta con la debida configuración para esta aplicación.



*Figura 129. Interfaz del Color Thresholder para la detección de la barra*

Como se puede observar, aunque la barra se distingue fácilmente por ser el principal objeto, es conveniente eliminar todo el ruido externo para así simplificar el algoritmo y que este sea más eficiente.

Para ello, se hará uso de la operación de morfología matemática de apertura y dilatación con un máscara rectangular y vertical respectivamente de la siguiente manera.

```
% Morfología matemática para eliminar imperfecciones y quedarnos con la
% línea vertical
se = strel("rectangle", [10, 1]);
se_dilate = strel("square", 3);
bw_opened = imopen(bw, se);
bw_opened = imdilate(bw_opened, se_dilate);
```

*Figura 130. Aplicación de la morfología matemática en la detección de la barra*

Tras aplicar esta parte del código el ruido se filtrará y la imagen pasará a ser la siguiente.



*Figura 131. Comparación para la morfología matemática en detectBar*

Como se puede ver, ya solo se tiene el objeto correspondiente con la barra de la señal más cercana, sin embargo, con el fin de añadirle robustez se prepara la detección para que en caso de haber más de un objeto se valoren todos y se elija el que sea más largo en el eje vertical.

Además, se plantea el caso en el que no se detecte ninguna barra, situación en la cual se devolverá un valor de infinito que no interfiera en el funcionamiento normal del vehículo. Otra medida de robustez incorporada es el if mediante el cual solo se tendrán en cuenta aquellos objetos con una orientación entre 80 y 100 grados y que a su vez, tengan más de 25 píxeles de longitud lo que eliminará posibles objetos residuales.

Con esto, se logra evitar errores ocasionados por falsas detecciones, a continuación se muestra el código responsable de estas implementaciones.

```
% Etiquetar las regiones conectadas en la imagen binaria
[L, numRegions] = bwlabel(bw_opened);

% Extraer propiedades de las regiones etiquetadas
props = regionprops(L, 'MajorAxisLength', 'Orientation');

% Ordenar las regiones por longitud del eje principal en orden descendente
[~, index] = sort([props.MajorAxisLength], 'descend');

% Iterar sobre las regiones etiquetadas
for i = 1:numel(index)

    % Longitud del eje principal de la región etiquetada
    barMajorAxisLength = props(index(i)).MajorAxisLength;

    % Ángulo de orientación de la barra
    angle = props(index(i)).Orientation;

    % Condiciones para detectar la barra
    if abs(angle) > 80 && abs(angle) < 100 && barMajorAxisLength > 25
        % Marcador de detección de la barra
        distanceToBar = (numPixV * barCurrentSize) / (2 * barMajorAxisLength * tand(AFOV / 2));
        barDetected = true;
        % Finalizar la iteración
        break;
    end
end

% Si no se detecta la barra, establecer la distancia como infinito
if ~barDetected
    distanceToBar = inf;
end
```

*Figura 132. Código para detectar la barra más cercana*

En el código de la figura anterior, se puede observar el algoritmo mediante el cual se obtiene la distancia hasta la barra una vez esta ha sido detectada. Este algoritmo, se basa en la técnica del pinhole en la cual conociendo el AFOV de la cámara, el tamaño real del objeto y el tamaño en píxeles del objeto y de la imagen se calcula la distancia real hasta el objeto.

Los parámetros estándar de la imagen y el objeto se introducen al principio de la función y junto con el esquema explicativo del pinhole en el cálculo de la distancia se pueden observar a continuación.



```
% Tamaño de la barra (metros) y parámetros de la cámara e imagen
barCurrentSize = 0.48978;
AFOV = 80;
numPixV = 256;
```

Figura 133. Parámetros para el cálculo de la distancia

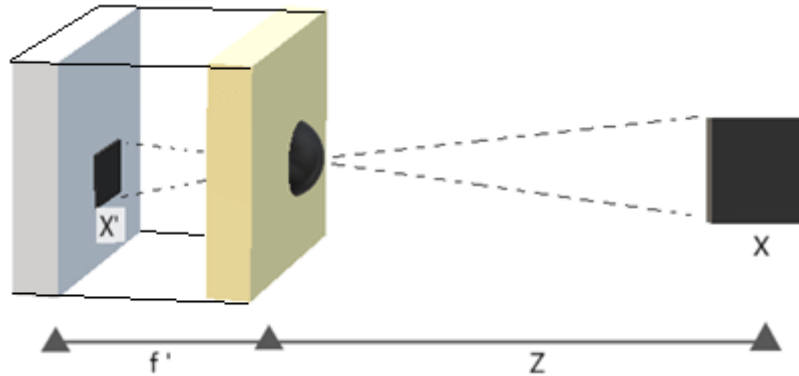


Figura 134. Esquema del pinhole [56]

Teniendo en cuenta el esquema anterior, la fórmula mediante la cual se obtiene la distancia real de la cámara al objeto es la siguiente.

$$\text{distancia real} = \frac{f * \text{tamaño barra (real)}}{\text{tamaño barra (imagen)}}$$

Donde la distancia focal (f) se calcula mediante la siguiente fórmula.

$$\text{distancia focal (f)} = \frac{\text{tamaño vertical de la imagen}}{2 * \tan(AFOV/2)}$$

Por lo tanto, la fórmula final que se ha implementado en el fragmento de código de la figura 130 será la que se muestra a continuación.

$$\text{distancia real} = \frac{\text{tamaño imagen (vertical)} * \text{tamaño barra (real)}}{2 * \text{tamaño barra (imagen)} * \tan(AFOV/2)}$$

Esta función auxiliar, se implementará en el script general de control y gracias a ella se podrán realizar funciones tales como la correcta detención ante un stop o la reducción de velocidad antes de un paso por curva, el código completo de la función utilizada para detectar la barra y la distancia a ella puede encontrarse en el anexo 1.11.

## 6. Algoritmo de planificación y seguimiento de trayectorias

En este capítulo, se desarrolla el último de los algoritmos de control del vehículo, este será el encargado de elegir la ruta óptima para ir desde la posición inicial del vehículo al destino deseado por el camino más corto según el conjunto de paths de la escena y sus conexiones.

Debido a la complejidad de este algoritmo, la implementación de esta funcionalidad se dividirá en varias secciones con sus debidas explicaciones detalladas sobre todas las funciones auxiliares creadas y la lógica detrás de estas. Todas estas funciones, se han almacenado siguiendo el esquema de la figura 20 en la carpeta de path e incluyen desde las funciones utilizadas para almacenar la información del entorno a las controladoras del vehículo en sí.

## 6.1. Obtención de las rutas

Lo primero que se debe hacer para poder obtener la trayectoria del vehículo, es conocer todos los componentes que forman la escena y todas las combinaciones disponibles que estas permiten. Cabe mencionar, que el conjunto de paths se ha obtenido de la escena de Coppelia con la ayuda de alguna de las funciones desarrolladas en el trabajo realizado sobre esta escena con anterioridad [20].

La primera función utilizada en este algoritmo de control es la función path, esta función, se encarga de almacenar en una misma estructura todas aquellas rutas de rectas, cruces, rotondas o parkings en una misma estructura con el siguiente formato.

```
paths =  
  
1×3 cell array  
  
{1×77 cell}    {1×162 cell}    {1×239 cell}
```

*Figura 135. Estructura de los paths del entorno*

E código de la función que ha logrado esto es el que se muestra a continuación.

```
function paths = path(streets_meta, roundabouts_meta, crossings_meta, parkings_meta)  
  
    paths = cell(1, 3);  
  
    % Adicción de las rutas de las calles  
    for i = 1:numel(streets_meta)  
        paths{1, 1} = [paths{1, 1}, create_paths_list(streets_meta{1, i}{1, 2})];  
    end  
  
    % Adicción de las rutas de las rotondas  
    for i = 1:numel(roundabouts_meta)  
        paths{1, 2} = [paths{1, 2}, create_paths_list(roundabouts_meta{1, i}{1, 3})];  
    end  
  
    % Adicción de las rutas de los cruces  
    for i = 1:numel(crossings_meta)  
        paths{1, 2} = [paths{1, 2}, create_paths_list(crossings_meta{1, i}{1, 2})];  
    end  
  
    % Adicción de las rutas de los parkings  
    for i = 1:numel(parkings_meta)  
        paths{1, 2} = [paths{1, 2}, create_paths_list(parkings_meta{1, i}{1, 2})];  
    end  
  
    paths{1, 3} = [paths{1, 1}, paths{1, 2}];  
end
```

*Figura 136. Código para la agrupación de los paths en una variable*

La función `create_path_list`, se ha creado para simplificar el código y que este sea más fácil de comprender, esta se encarga de almacenar cada tipo de path en la variable conjunta `paths` sobre la cual se trabajará en el futuro, esta función se adjunta a continuación.

```
function paths_list = create_paths_list(raw_meta_list)
    paths_list = {};
    for i = 1:numel(raw_meta_list)
        raw_meta = raw_meta_list{i};
        path = createSimPath(raw_meta);
        paths_list(end+1) = path;
    end
end
```

**Figura 137. Función `create_path_list`**

La función `createSimPath` es la encargada de clasificar y ordenar la información de cada ruta de manera que se pueda acceder a cada característica individual como por ejemplo el punto inicial o final o el nombre como se puede observar en la siguiente figura.

```
function simPath = createSimPath(raw_meta)
    simPath = struct();
    simPath.puntos = {};
    simPath.start_point = raw_meta{1, 5}{1, 1};
    for i = 1:numel(raw_meta{1, 5})
        simPath.puntos(end+1) = raw_meta{1, 5}{1, i};
    end
    [~, numCols] = size(raw_meta{1, 5});
    simPath.end_point = raw_meta{1, 5}{1, numCols};
    simPath.length = raw_meta{3};
    simPath.name = raw_meta{2};
    simPath.signs = createSignList(raw_meta{4});
    simPath.successors = {}; % Inicializar como una celda vacía
    simPath.predecessors = {}; % Inicializar como una celda vacía
end
```

**Figura 138. Función `createSimPath`**

Cabe destacar que la variable `signs` se ha obtenido de manera indirecta mediante la función externa `createSignList` que se adjunta a continuación.

```
function signList = createSignList(raw_meta)
    signList = {};
    for i = 1:numel(raw_meta)
        sign = createSign(raw_meta{i});
        signList(end+1) = sign;
    end
end
```

**Figura 139. Función `createSignList`**

A su vez, la última función utilizada dentro de esta es `createSign` y que tendrá como objetivo y sentido lo mismo que las mencionadas con anterioridad y que se puede observar en la siguiente figura.

```
function sign = createSign(raw_meta)
    sign = struct();
    sign.type = raw_meta{1};
    sign.position = raw_meta{2};
end
```

Figura 140. Función createSign

## 6.2. Clasificación de las rutas

Una vez todas las rutas han sido obtenidas de manera ordenada y detallada, estas se clasificarán según su conexionado, para ello, se mide la distancia entre el punto inicial y final de cada ruta con todas las demás. En caso de que esta distancia sea menor de 0.01, valor arbitrario decidido en concordancia con la escena. Estas rutas, se almacenarán en las variables Origen y Destino junto con la longitud de este camino en weights.

Este cálculo de la distancia se realiza mediante la obtención del módulo de la diferencia entre las coordenadas x e y de ambos punto, a continuación, se adjunta un esquema explicativo de este cálculo junto con la función utilizada con esta finalidad.

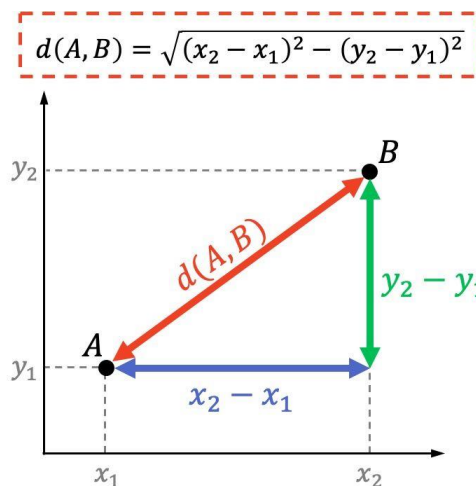


Figura 141. Esquema cálculo de distancia [57]

```
function distance = getDistance(point1, point2)
    distance = sqrt(((point2{1} - point1{1})^2) + ((point2{2} - point1{2})^2));
end
```

Figura 142. Función auxiliar para el cálculo de la distancia

De esta forma, se consigue relacionar las rutas con sus vecinos según la distancia entre estos, lo que será fundamental para la posterior implementación del planificador de ruta. Todo este trabajo, se realiza mediante la función auxiliar clasi\_paths, que puede encontrarse a continuación junto con las explicaciones necesarias.

```
function [Origen, Destino, weights] = clasi_paths(paths, ACCEPTABLE_POINTS_DISTANCE)
N =0;

% Inicio de los bucles anidados para el análisis de todas las combinaciones
for i = 1:numel(paths{1, 3})
    p1 = paths{1, 3}{i};
    for j = 1:numel(paths{1, 3})
        p2 = paths{1, 3}{j};
        % Si distancia de p1 a p2 es menor de 0.01
        if getDistance(p1.end_point, p2.start_point) < ACCEPTABLE_POINTS_DISTANCE
            N = N+1;
            Origen(N) = i;
            Destino(N) = j;
            weights(N) = p2.length;

        end
        % Si distancia de p2 a p1 es menor de 0.01
        if getDistance(p1.start_point, p2.end_point) < ACCEPTABLE_POINTS_DISTANCE
            N = N+1;
            Destino(N) = i;
            Origen(N) = j;
            weights(N) = p1.length;

        end
    end
end
end
end
```

*Figura 143. Función auxiliar para la clasificación de las rutas*

Para comprobar que este proceso ha sido correcto y no se ha quedado ninguna ruta aislada que sea inalcanzable, se crea un grafo con todas aquellas conexiones posibles. Además, para facilitar la comprensión del algoritmo, se crearán dos figuras de tal forma que en una de ellas puedan observarse los pesos de estas combinaciones (distancias) pero no en la segunda.

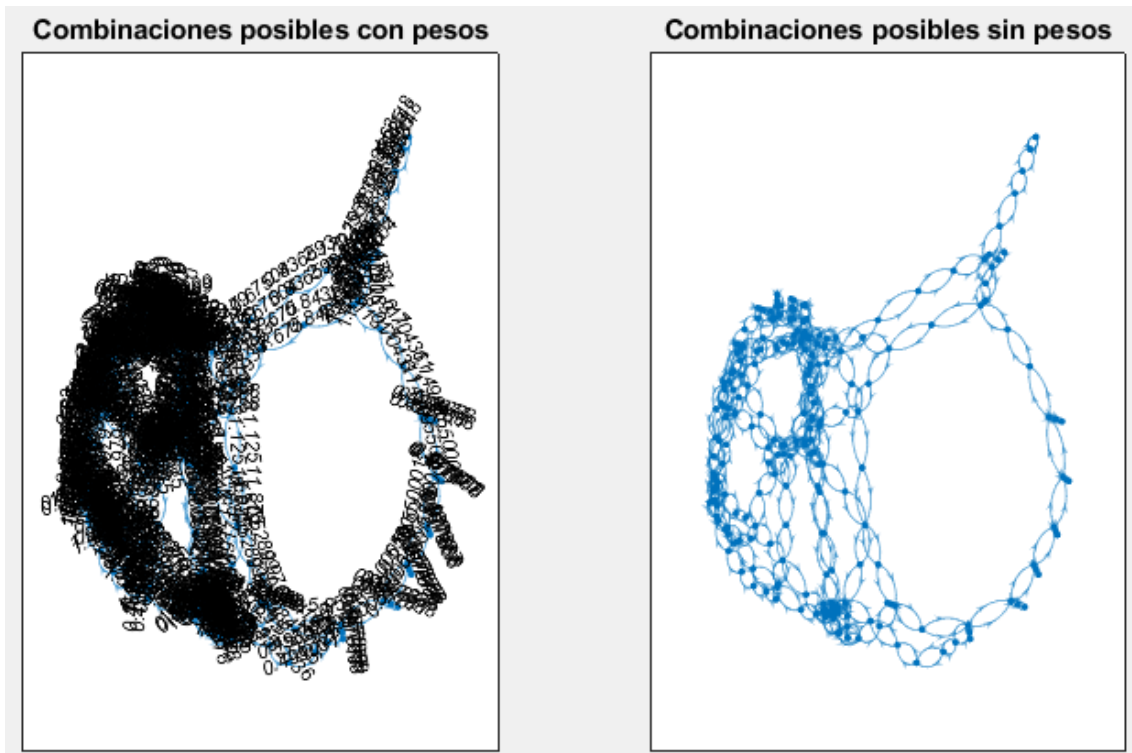
El código utilizado para ello junto con las imágenes resultantes se puede encontrar a continuación.

```
% Crear el gráfico con las combinaciones y los pesos
G = digraph(Origen, Destino, weights);

% Primer subplot: gráfico con pesos en las aristas
subplot(1, 2, 1);
plot(G, 'EdgeLabel', G.Edges.Weight);
title('Combinaciones posibles con pesos');

% Segundo subplot: gráfico sin pesos en las aristas
subplot(1, 2, 2);
plot(G);
title('Combinaciones posibles sin pesos');
```

*Figura 144. Código para mostrar los posibles caminos*



*Figura 145. Grafos del conexionado de las rutas del entorno*

Como se puede observar, hay un gran número de combinaciones posibles que deberán ser comprobadas de cara a la obtención de la ruta más corta, además, no hay ninguna ruta aislada por lo que el vehículo podrá ir a cualquiera de estas sin problemas.

### 6.3. Obtención de la información

Una vez se tienen ya todas las rutas procesadas y clasificadas, se podrá empezar a trabajar con estas para obtener la ruta más corta entre dos puntos, para ello, la primera parte será la obtención de la ruta origen y destino.

Ambos procedimientos se desarrollan en los siguientes subapartados de manera individual con sus debidas funciones auxiliares.

#### 6.3.1. Ruta de origen

Para obtener la ruta de origen, se ha diseñado un algoritmo mediante el cual se obtiene la ruta más cercana al vehículo utilizando el GPS de este. Para ello, se calcula la distancia entre el vehículo y los puntos de cada ruta de manera análoga al procedimiento de las figuras 139 y 140, la ruta que incluya el punto con la menor distancia hasta el coche se utilizará como punto de partida.

El fragmento de la función auxiliar mediante la cual se obtiene el punto más cercano al vehículo y por lo tanto la ruta de origen desde la que partirá el algoritmo se adjunta a continuación.

```
% Iterar sobre cada ruta en el elemento
for j = 1:numel(rutas)
    % Obtener los puntos de la ruta actual
    puntos = rutas{j}.puntos;

    % Calcular la distancia entre el GPS y cada punto en la ruta
    distancias = zeros(numel(puntos), 1);
    for k = 1:numel(puntos)
        punto = puntos{k};
        distancias(k) = sqrt((punto{1} - GPS{1}).^2 + (punto{2} - GPS{2}).^2);
    end

    % Encontrar la distancia mínima y el índice correspondiente
    [dist_min_ruta, idx] = min(distancias);

    % Actualizar la distancia mínima y la ruta más cercana si es necesario
    if dist_min_ruta < distancia_minima
        distancia_minima = dist_min_ruta;
        ruta_cercana = rutas{j};
        ruta_cercana.distancia_minima = dist_min_ruta;
        ruta_cercana.punto_mas_cercano = puntos{idx};
    end
end
```

*Figura 146. Función auxiliar para la obtención del punto de partida*

### 6.3.2. Ruta de destino

Para elegir la ruta destino a la que se debe llegar, se ha creado una función mediante la cual el usuario puede elegir el lugar al que se desea ir. En este caso, se ha optado por una implementación mediante la cual el usuario debe introducir por texto el destino deseado, aunque existen otras alternativas como la selección por pantalla clicando en el lugar requerido.

Sin embargo, se ha decidido la opción escrita ya que es la más sencilla y agradable de cara al usuario y la más escalable para futuras implementaciones. Para la obtención de este destino, se solicita al usuario que escriba el destino deseado, una vez este ha sido escrito, se comprueban todas las rutas posibles para en caso de existir pasar a la siguiente etapa y en caso negativo, solicitar al usuario introducir otra vez la ruta hasta que el nombre de esta sea correcto.

La función auxiliar que implementará esta funcionalidad, y devolverá el nombre de la ruta de manera análoga a la de origen, se adjunta a continuación.

```
function destino = obtenerDestinoValido(paths)
% Solicitar al usuario que introduzca el nombre del path destino o "end" para terminar
while true
    nombre_destino = input('Introduce el nombre del path destino (por ejemplo, path163), o escribe "end" para terminar: ', 's');

    % Verificar si el usuario desea terminar la simulación
    if strcmp(nombre_destino, 'end')
        disp('Terminando la simulación. ');
        destino = [];
        break; % Salir del bucle
    end

    % Verificar si el nombre del path destino existe
    indice_destino = getInfo(nombre_destino, paths);
    if ~isempty(indice_destino)
        disp('Destino válido. ');
        destino = paths{1, 3}{indice_destino}.name;
        break; % Salir del bucle
    else
        disp('El destino no existe. Por favor, introduce otro destino. ');
    end
end
end
```

*Figura 147. Obtención de la ruta destino*

Cabe mencionar, la implementación de un algoritmo que permita al usuario salir de este bucle mediante la escritura de la palabra “end”, esto será útil para incrementar la robustez del algoritmo y evitar bucles infinitos que puedan dejar el sistema inoperativo.

### 6.3.3. Cálculo de la trayectoria más corta

Para el cálculo de la trayectoria más cercana entre la posición actual del vehículo y el destino al que el usuario ha seleccionado, lo primero que se hará es obtener la información de ambas posiciones o rutas.

Teniendo en cuenta que las funciones encargadas de recabar esta información solo devolvían el nombre en formato de string, se ha creado una función auxiliar para obtener toda la información de estas rutas. Esta función, comprueba el nombre de las rutas de origen y destino y las compara con el conjunto hasta que coincida con alguna, caso en el que se para la ejecución de esta función y devuelve toda la información de esta ruta en particular.

A continuación, se adjuntan las figuras del código de la función auxiliar junto con su implementación en el script principal.

```
function info = getInfo(string, paths)
    info = [];

    for j = 1:numel(paths{1, 3})
        iteracion = paths{1, 3}{j}.name;
        % Si coinciden sal del bucle
        if strcmp(iteracion, string)
            info = j;
            break
        end
    end
end
```

*Figura 148. Función auxiliar para la obtención de la información de origen / destino*

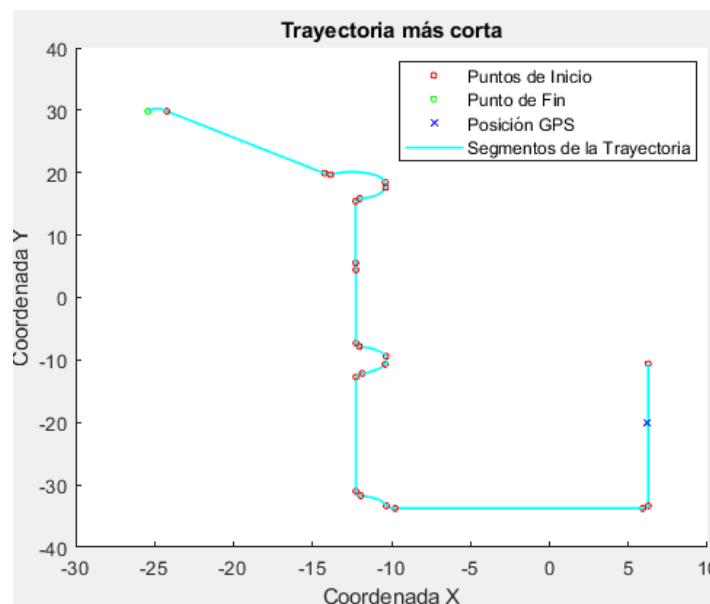


```
PathOrigen = getInfo(RutaOrigen, paths);  
PathDestino = getInfo(RutaDestino, paths);
```

*Figura 149. Implementación de la función getInfo*

Para la obtención de la ruta más corta entre el origen y el destino, se ha utilizado la función de Matlab *shortestpath* [58] la cual tiene como entradas el grafo de la figura 143 junto con el origen y el destino entre los que se quiere obtener la ruta más corta.

Una vez esta ruta se ha obtenido, se ha creado una función para mostrar una imagen con la trayectoria completa del recorrido junto con la posición inicial del vehículo. Aunque el código completo puede encontrarse en el anexo 1.12, se adjunta una imagen resultante de aplicar la función *shortestpath*.

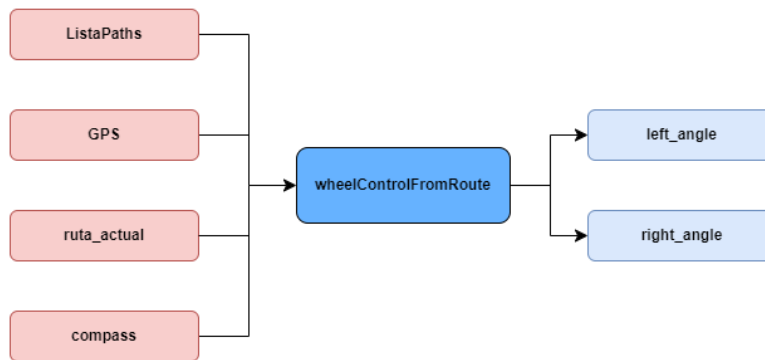


*Figura 150. Imagen ejemplificativa de la trayectoria*

Además, esta función servirá para comprobar la eficiencia del algoritmo de control ya que se podrá dibujar la trayectoria del vehículo sobre la ideal y comprobar la eficiencia de este procesamiento. Una vez se ha obtenido la ruta más corta entre el origen y el destino, ya se podrá pasar al algoritmo para el seguimiento de esta lo que se detallará en el siguiente punto.

## 6.4. Seguimiento de la trayectoria

Primero, se muestra una figura con la cabecera de la función utilizada donde se muestren las entradas y las salidas de esta función para saber qué es lo que se espera conseguir con esta.



**Figura 151. Esquema de la cabecera de la función auxiliar wheelControlFromRoute**

De manera análoga al control que se realizó para el seguimiento del carril, se ha desarrollado una función para el seguimiento de la trayectoria por gps. En esta, se plantea primeramente un sistema similar al explicado en la función para obtener el punto de partida mediante el siguiente código.

```

% Obtener los puntos de la ruta más cercana
puntos = ruta_actual.puntos;

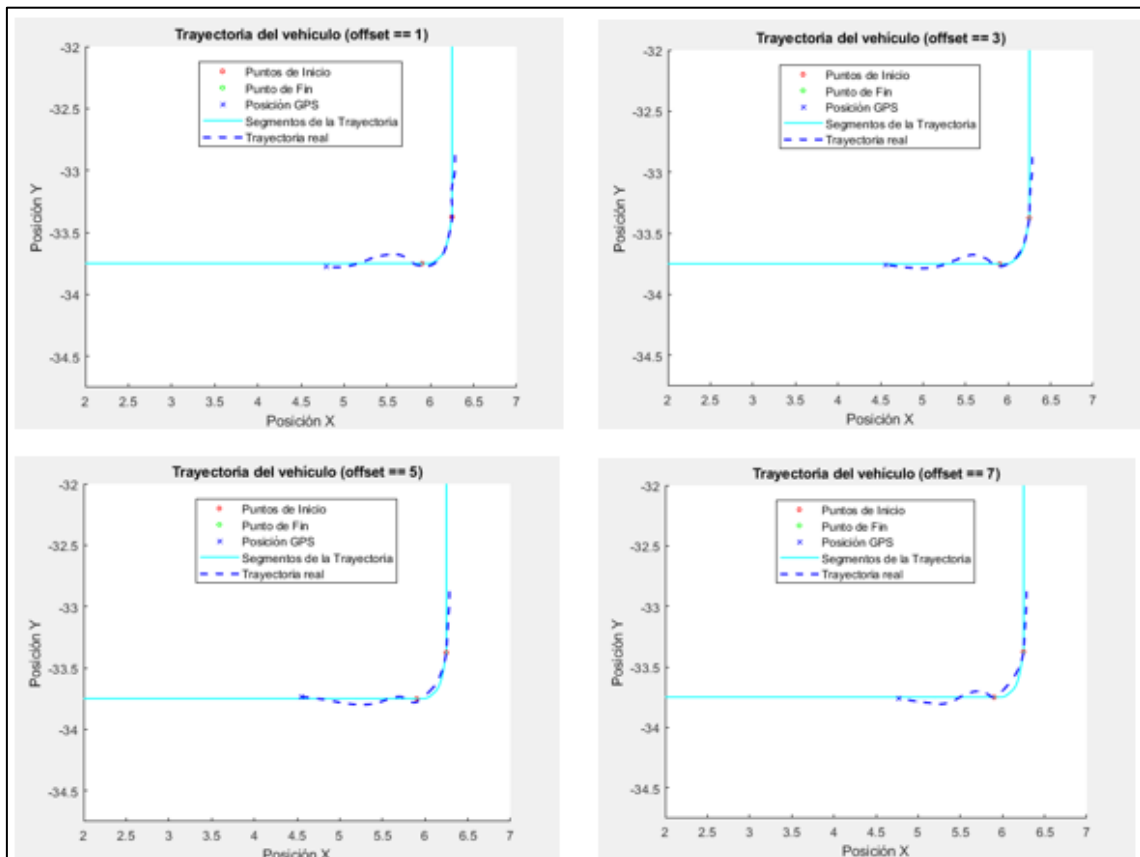
% Inicializar la distancia mínima y el índice del punto más cercano
distancia_minima = Inf;
indice_punto_cercano = 0;

% Calcular la distancia entre el GPS y cada punto en la ruta
for j = 1:length(puntos)
    punto = puntos{j};
    distancia = sqrt((punto{1} - GPS{1})^2 + (punto{2} - GPS{2})^2);
    if distancia < distancia_minima
        distancia_minima = distancia;
        indice_punto_cercano = j;
    end
end
end
  
```

**Figura 152. Obtención del punto más cercano**

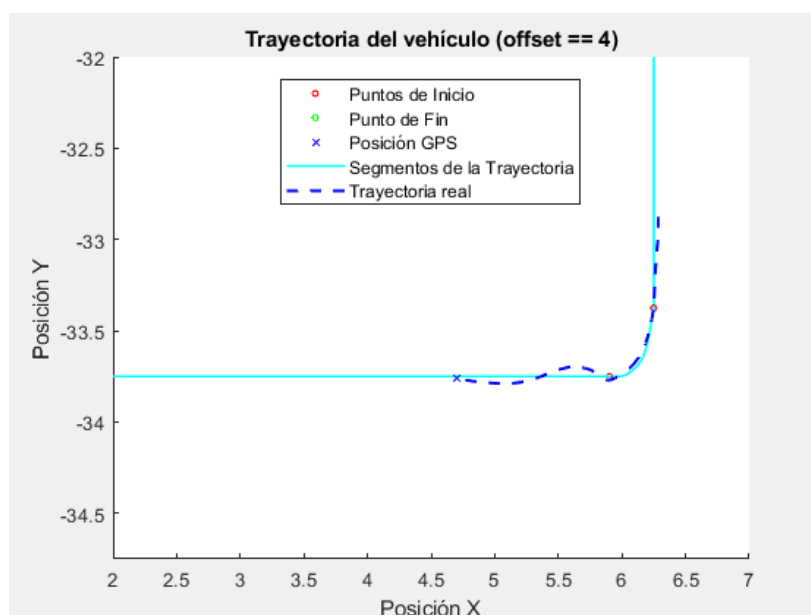
Una vez se ha obtenido el punto más cercano al vehículo se ha decidido no utilizar este como referencia ya que podría dar problemas en caso de que este estuviese por detrás del coche o no lo suficientemente lejos. Además, al tomar como punto de referencia uno más lejano, el coche podrá adaptarse a este con mayor suavidad y antelación lo que es muy importante a la hora de conducir un vehículo.

Para la selección de este punto, incremento entre el punto más cercano y referencia, se han realizado varias pruebas para ver cual ofrece un mejor seguimiento de la trayectoria en el paso por curva. A continuación, se adjunta las figuras resultantes tras aplicar estas pruebas.



**Figura 153. Pruebas para la selección del punto referencia**

Como se puede observar en la figura anterior, el mejor paso por curva es con un offset de 3 puntos aunque su conversión al modo de siguelineas en la recta no ha sido tan suave. Esta transición, se suaviza enormemente con un offset de 4 por lo que buscando un punto medio entre ambas el offset final será de 4, devolviendo el siguiente seguimiento de trayectoria.



**Figura 154. Seguimiento de la trayectoria con un offset de 4**

Además, este offset solo se aplicará en caso de no ser el punto referencia uno de los 4 últimos de la ruta, caso en el que no se aplicará ningún offset, la implementación de este offset se realiza mediante las siguientes líneas de código.

```
% Asegurarse de que el índice objetivo no exceda el número de puntos
indice_punto_objetivo = min(indice_punto_cercano + 4, length(puntos));
punto_objetivo = puntos{indice_punto_objetivo};
```

*Figura 155. Implementación del punto objetivo*

Posteriormente, se obtiene la diferencia de las coordenadas x e y del coche con las del punto objetivo, y se calcula la arco tangente de cuatro cuadrantes de estas diferencias para obtener el ángulo del coche respecto a este. Cabe destacar, que para obtener el cálculo del ángulo objetivo, se debe restar la orientación del vehículo a esta arco tangente mediante el código mostrado a continuación.

```
% Obtener la diferencia en x y y entre el GPS y el punto objetivo
x_diff = punto_objetivo{1} - GPS{1};
y_diff = punto_objetivo{2} - GPS{2};

% Calcular el ángulo entre el GPS y el punto más cercano
angle = atan2(y_diff, x_diff) - compass;
angle = rad2deg(angle);
```

*Figura 156. Cálculo del ángulo objetivo*

Una vez se ha obtenido este, se aplica el procedimiento para el control de un vehículo Ackerman que ya se planteó en el subapartado 5.1.7.1. y más en particular en la figura 117, esta parte de la función se muestra a continuación.

```
if angle ~= 0
    radius = LENGTH / tan(deg2rad(angle));
else
    radius = Inf;
end

close_angle = atan(LENGTH / (abs(radius) - WIDTH / 2));
far_angle = atan(LENGTH / (abs(radius) + WIDTH / 2));

if radius > 0
    % Giro hacia la izquierda
    left_angle = far_angle;
    right_angle = close_angle;
else
    % Giro hacia la derecha
    left_angle = -close_angle;
    right_angle = -far_angle;
end
```

*Figura 157. Control del vehículo para el algoritmo del GPS*

El código completo de esta función auxiliar, donde se integran todas las partes que han sido explicadas en este subapartado con anterioridad, se adjunta en el anexo 1.13.

## 6.5. Transmisión de los datos

Para la transmisión de los datos al vehículo una vez se han obtenido los ángulos de giro, se usará la función ya utilizada con este fin en el seguimiento del carril y que se adjunta a continuación.

```
function setCarControl(sim, steer, lights, speed, leftAngle, rightAngle, lightOption)
    steer{2}{1} = leftAngle;
    steer{2}{2} = rightAngle;
    steer{1} = speed;
    lights{1} = lightOption;
    data = {steer, lights};
    sim.callScriptFunction('set_state@/Car',sim.scripttype_childscript, data);
end
```

*Figura 158. Función para la transmisión de los datos en el algoritmo del GPS*

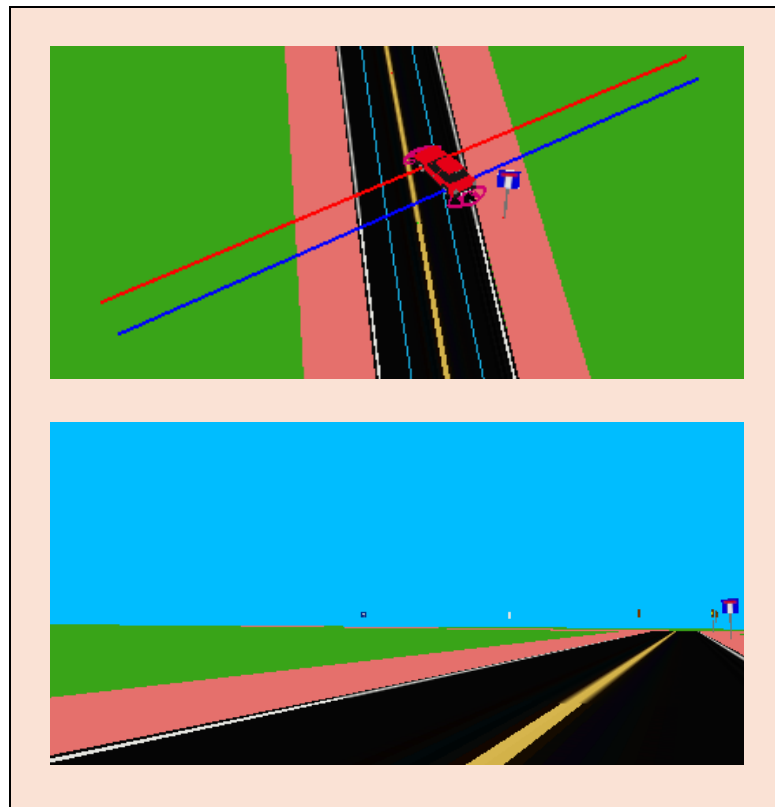
La reutilización de la misma función, se ha logrado mediante el uso de la misma arquitectura en ambos algoritmos, de esta forma, se consigue un trabajo mucho más compacto y uniforme ya que en ambos casos, se plantea el algoritmo para dar las mismas salidas independientemente de cuales sean las entradas.

## 6.6. Pruebas del algoritmo

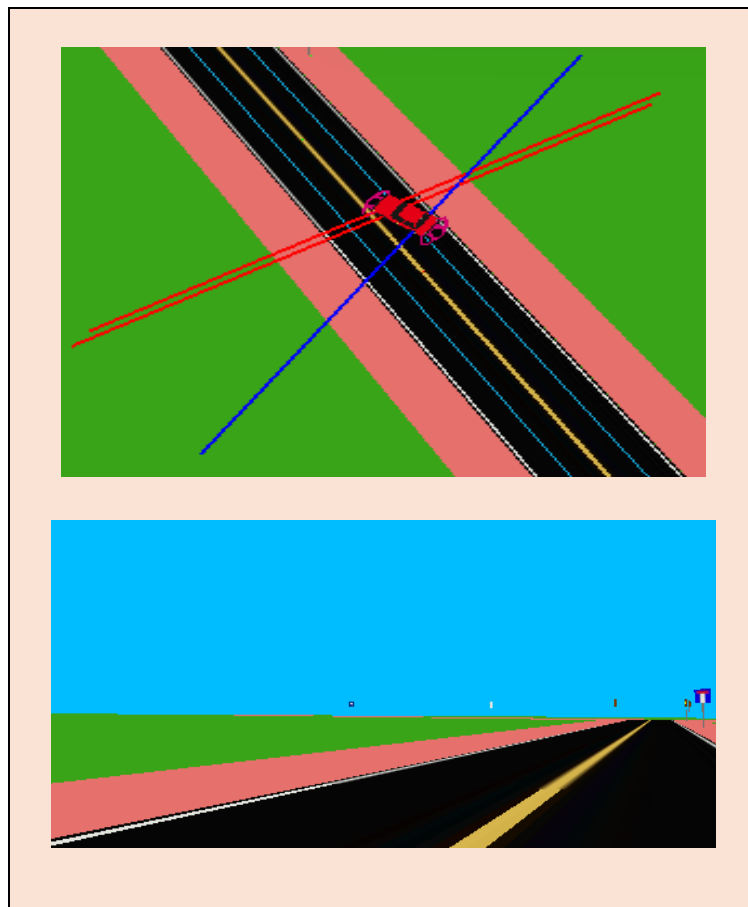
Al igual que se realizó en los algoritmos de los apartados anteriores, se han realizado pruebas de estrés que pongan a este en situaciones límite de las cuales este deba salir sin mayores complicaciones. Para realizar estas pruebas se ha creado un script exclusivo para realizar estos en el cual el control por GPS sea el único y principal algoritmo de control, este script se puede encontrar de manera íntegra en el anexo 1.14. Las pruebas de estrés realizadas para probar la eficacia de este algoritmo serán similares a las del algoritmo de carril pero con su debida adaptación para centrarlo en los puntos débiles que deberá superar este algoritmo, estas pruebas se desarrollan en los siguientes subapartados.

### 6.6.1. Reconducción de la trayectoria

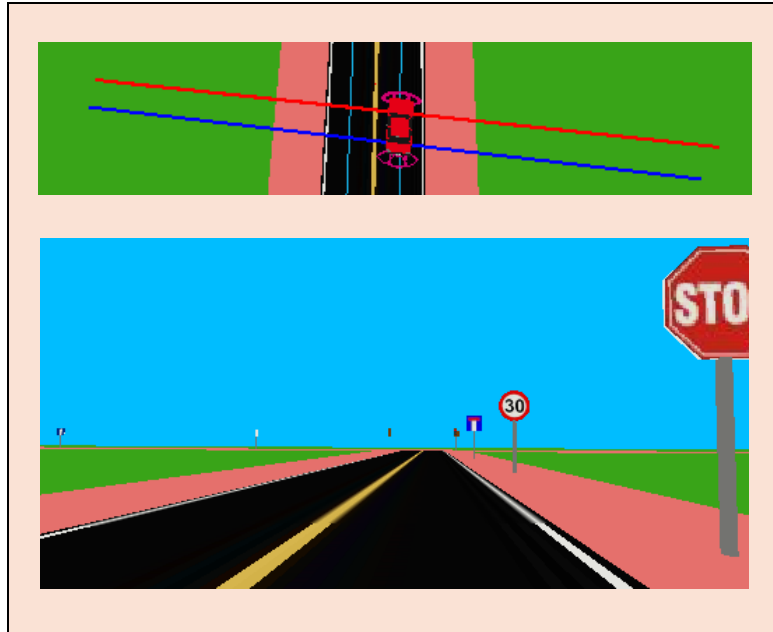
En esta prueba, el vehículo se encontrará desorientado respecto de la trayectoria y deberá girar suavemente hasta reconducirse y situarse encima de los puntos objetivo. A continuación, se adjuntan 3 imágenes obtenidas durante la realización de este.



*Figura 159. Punto inicial de la prueba*



*Figura 160. Instantánea tomada durante el test*

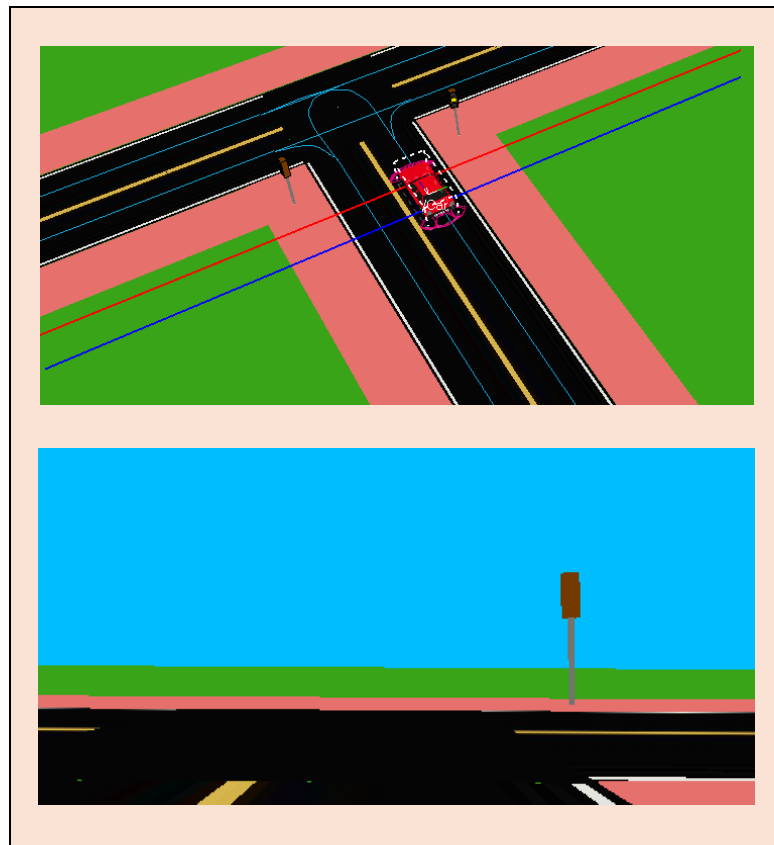


*Figura 161. Reconducción completada*

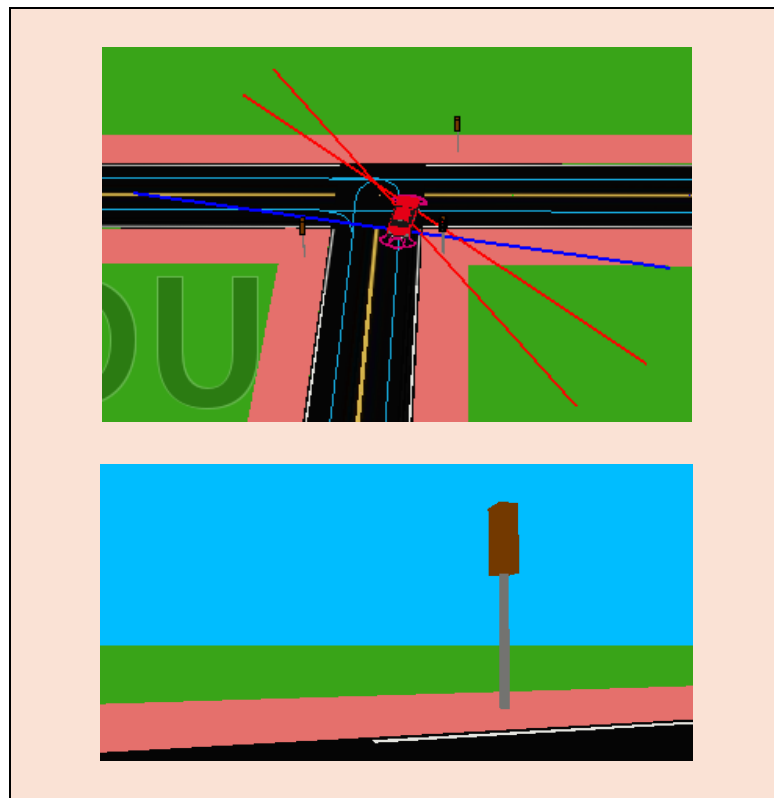
Como se puede ver, el test ha sido superado y el vehículo, a pesar de estar claramente girado, es capaz de recuperar la orientación y sentido correcto y corregir este problema.

### **6.6.2. Paso por curva**

La segunda prueba de estrés que se ha realizado para este algoritmo, es su adaptación a la trayectoria de los cruces, para probar esta situación, se somete al vehículo al peor de los casos posibles. Esta situación de máximo riesgo, se dará cuando el vehículo tome la curva más cerrada posible en un cruce y por lo tanto esa será la situación que se recrea en este test de estrés.

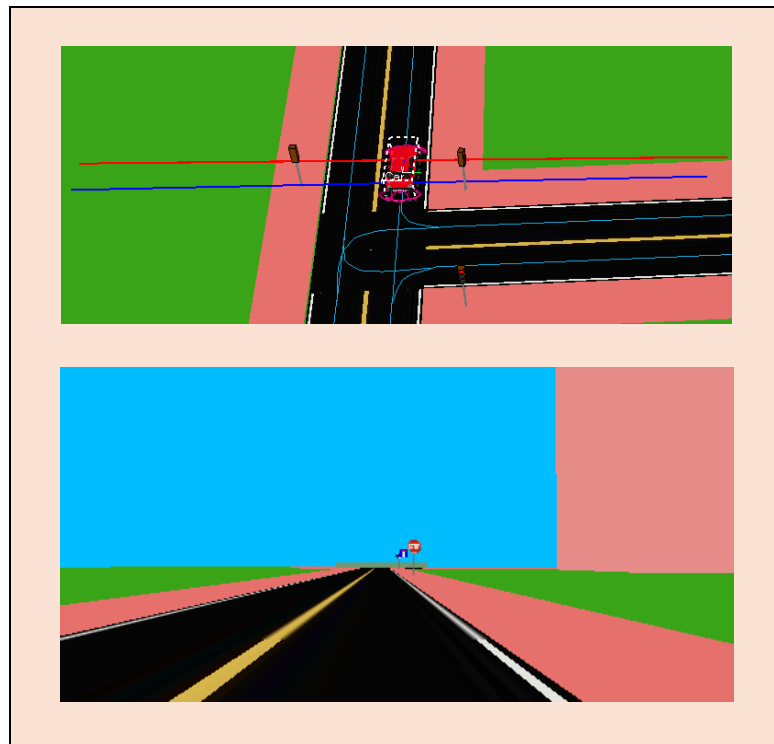


*Figura 162. Situación inicial antes del cruce*



*Figura 163. Instantánea tomada durante la curva*





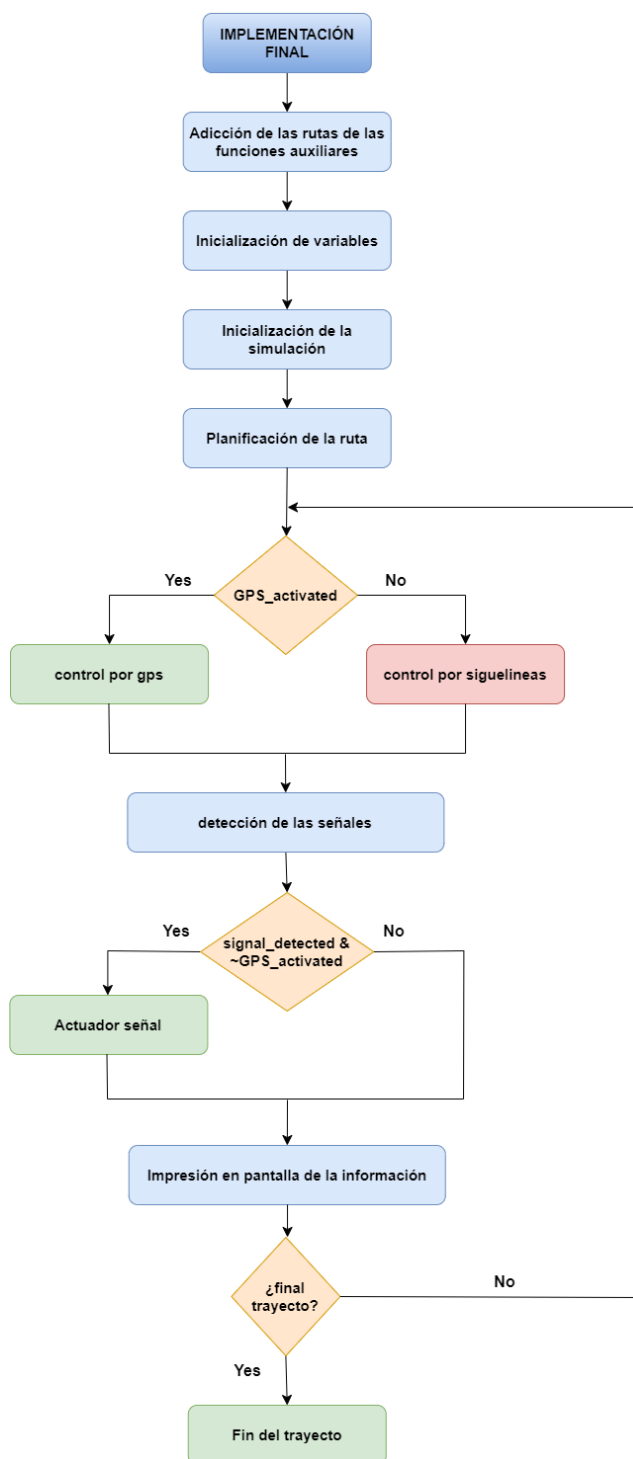
*Figura 164. Imagen del vehículo después de girar*

Como se puede observar, esta prueba también ha sido superada y por lo tanto se considerará este algoritmo apto para la posterior implementación en el sistema final.

## 7. Implementación final

Una vez se han implementado todos los módulos del sistema tal y como se mostraba en el esquema de la figura 1, se procede con la implementación de todos ellos en un mismo modelo que permita controlar el vehículo en todo momento de la manera más eficiente posible. Cabe recordar, que en multitud de escenarios más de un sistema sería capaz de continuar con la ejecución sin problemas pero al ser la conducción autónoma un campo tan delicado es importante la redundancia de cara a aumentar la robustez.

Para decidir qué sistema será el principal en cada momento, se seguirá el esquema lógico que se adjunta a continuación en cada iteración.



**Figura 165. Esquema del control completo del vehículo**

Aunque en los capítulos anteriores se han detallado las características de cada una de las partes del algoritmo de control, a lo largo de este capítulo se desarrolla el script principal y como se ha implementado de manera ordenada todo lo mencionado con anterioridad. El código completo se puede encontrar en el anexo 1.15 aunque en los siguientes apartados se explica este detalladamente.

## 7.1. Inicialización

En este apartado, se explicará el conjunto de inicializaciones requeridas antes de comenzar a trabajar sobre el vehículo. Esta parte del script se encuentra fuera del bucle principal y se podrá dividir en los subapartados que se observan en la figura 149 y que se explican en los siguientes puntos.

### 7.1.1. Adición de las funciones auxiliares

Antes de comenzar con la ejecución, es necesario reiniciar el sistema, para evitar posibles errores derivados de ejecuciones anteriores, e incluir el conjunto de funciones auxiliares que se han ido desarrollando a lo largo del trabajo.

A continuación se muestra la parte del script donde se realiza este trabajo.

```
%% -----IMPLEMENTACIÓN CONJUNTA-----  
clear all;  
close all;  
  
addpath('Function/Carril/');  
addpath('Function/Señales/');  
addpath('Function/Path/');  
% -----
```

*Figura 166. Reinicio del sistema y adición de las funciones auxiliares*

### 7.1.2. Inicialización de las variables

Una vez el sistema se ha reiniciado y se han incluido las rutas globales necesarias, se procede con la declaración de las variables utilizadas tanto de manera local como externa por las funciones auxiliares. Además, se carga la red que se ha entrenado con este propósito y se almacena en la estructura “red”.

Esta parte de la inicialización se realiza mediante la siguiente parte del código.

```
%% ----- VARIABLES -----  
signal_detected = false;  
% doble_filtro = 0;  
% status = 'recta';  
detection_text = "No se detectó ninguna señal.";  
red = load('Deep Learning/Resnet50_mvaltor.mat');  
speed = 0.6;  
lightOption = 1;  
time_started = 0;  
YPred_ant = '';  
debounce = 0;  
  
LENGTH = 0.316;  
WIDTH = 0.213;  
ACCEPTABLE_POINTS_DISTANCE = 0.01;  
% -----
```

*Figura 167. Inicialización de las variables necesarias*

### 7.1.3. Inicialización de la simulación

Posteriormente, se ha procedido a la inicialización de la simulación y la obtención de los parámetros iniciales de esta, tales como el GPS, el tiempo, la orientación, el sensor de visión, etcétera.

En la siguiente figura se muestran el conjunto de funciones que tienen este propósito.

```
%----- INICIALIZACION SIMULACION-----  
  
client = RemoteAPIClient();  
sim = client.getObject('sim');  
fprintf('Program started\n');  
  
visionSensorHandle = sim.getObject('/ViewCamera');  
passiveVisionSensorHandle = sim.getObject('/ViewVisualization');  
  
client.setStepping(true);  
sim.startSimulation();  
  
startTime = sim.getSimulationTime();  
t = startTime;  
client.step();  
  
[gps, compass, steer, lights] = sim.callScriptFunction('get_state@/Car',sim.scripttype_childscript);  
meta = sim.callScriptFunction('get_meta@/Meta',sim.scripttype_childscript);  
%-----
```

*Figura 168. Inicialización de la simulación*

### 7.1.4. Inicialización del seguimiento de la trayectoria

Una vez se ha inicializado correctamente la simulación, ya se podrá pasar a la última etapa de la inicialización, la obtención de la ruta planeada. Para ello, se siguen los pasos explicados en el capítulo anterior para obtener la ruta que se debe seguir y comenzar con el algoritmo de esta.

En la siguiente figura, se muestran las funciones utilizadas con este propósito.

```
% ----- INICIALIZACION PATH -----  
  
[roundabouts_meta, streets_meta, crossings_meta, parkings_meta] = meta{:};  
paths = path(streets_meta, roundabouts_meta, crossings_meta, parkings_meta);  
  
[Origen, Destino, weights] = clasi_paths(paths, ACCEPTABLE_POINTS_DISTANCE);  
  
% Crear el gráfico con las combinaciones y los pesos  
G = digraph(Origen, Destino, weights);  
  
% Primer subplot: gráfico con pesos en las aristas  
subplot(1, 2, 1);  
plot(G, 'EdgeLabel', G.Edges.Weight);  
title('Combinaciones posibles con pesos');  
  
% Segundo subplot: gráfico sin pesos en las aristas  
subplot(1, 2, 2);  
plot(G);  
title('Combinaciones posibles sin pesos');  
  
RutaOrigen = obtenerOrigen(paths, gps);  
RutaDestino = obtenerDestino(paths);  
  
PathOrigen = getInfo(RutaOrigen, paths);  
PathDestino = getInfo(RutaDestino, paths);  
  
[P,d] = shortestpath(G, PathOrigen, PathDestino);  
ListaPaths = getPaths(P, paths);  
  
ruta_seleccionada(ListaPaths, gps);  
  
% -----
```

*Figura 169. Obtención de la ruta planeada*

Una vez se han aplicado todos estos preparativos, se podrá pasar con la ejecución del seguimiento de la trayectoria deseada y que se explica en el siguiente apartado.

## 7.2. Seguimiento de la trayectoria

Llegados a este punto, ya se tendrá toda la información necesaria para iniciar el seguimiento de la trayectoria por lo que se podrá llamar al bucle principal, este, ejecutará de manera periódica el algoritmo reflejado en la figura 158 hasta que el vehículo llegue a su destino. Lo primero que se deberá hacer dentro de este bucle, es obtener la imagen que se está captando desde el vehículo lo que se hará mediante la siguiente función.

```
function img = getImage(sim, handle)  
    [img, resX, resY] = sim.getVisionSensorCharImage(handle);  
    img = flip(permute(reshape(img, 3, resX, resY), [3 2 1]), 1);  
end
```

*Figura 170. Obtención de la imagen desde Coppelia*

Posteriormente, el control deberá elegir entre utilizar el algoritmo del seguimiento de carril o bien el seguimiento de la trayectoria, en este proyecto, se ha decidido que el seguimiento por GPS solo se utilice para cruces o rotondas mientras que el de seguimiento del carril será el principal.

Es por ello, que la variable GPS\_activated se ha inicializado como falso y no será activada hasta que el algoritmo de detección del carril no detecte ambas líneas del carril y por lo tanto esté en unos de los escenarios anteriormente mencionados.

La implementación de esta parte del código se muestra a continuación.

```

if GPS_activated
    % Obtener la ruta más cercana
    ruta_actual = ruta_cercana(ListaPaths, gps);

    % Se reduce la velocidad por ser un tramo de curvas
    speed = 0.2;
    [leftAngle, rightAngle] = wheelControlFromRoute(ListaPaths, gps, ruta_actual, compass);
    [~, ~, ~, bordes_carril, GPS_activated] = deteccion_carril_mvalor(img);
    modo = 'GPS';
else
    [laneLeft, laneRight, centerLaneRight, bordes_carril, GPS_activated] = deteccion_carril_mvalor(img);
    [barDetected, distanceToBar] = detectBar(img);
    [leftAngle, rightAngle] = wheelControl(laneLeft, laneRight);
    speed = aceleracion(laneLeft, laneRight);
    modo = 'Siguelíneas';
end

```

*Figura 171. Selección del algoritmo para el seguimiento de la trayectoria*

En caso de haberse detectado ambas líneas del carril se activará el seguimiento del carril y la detección de la barra de las señales mientras que en caso de no haber sido detectadas se entrará en el modo de GPS. En este modo, no será necesario el cálculo de la distancia respecto de la barra ya que al estar en un cruce o una rotonda no será necesario detectar señales y por lo tanto se agilizará la ejecución de esta forma.

Cabe destacar, la llamada a la función de clasificación del carril desde el modo GPS ya que esta será la encargada de decidir en que momentos se debe cambiar de modo. A su vez, servirá para obtener la imagen capturada para enviarla posteriormente a Coppelia y que el usuario vea en tiempo real el procesamiento que se está realizando.

Una vez se ha seleccionado el algoritmo utilizado para el seguimiento de la trayectoria, se continua con la siguiente decisión que el modelo deberá tomar. En caso de haberse detectado una señal que cumpla con la lógica que se explicó a lo largo del capítulo 4 y estar en modo seguilineas, por lo explicado con anterioridad, se llama a la función actuador\_signal mientras que en caso negativo no será necesario realizar esta llamada.

Aunque esta función podría llamarse en cada iteración debida a las medidas de robustez aplicadas en esta, al no llamarse siempre se consigue un algoritmo más rápido y eficaz, a continuación, se adjunta el bucle condicional para dicha implementación.

```

[debounce, detection_text, YPred, YPred_ant, signal_detected] = deteccion_signal_mvalor(YPred_ant, ...
    img, red, signal_detected, detection_text, debounce);
if (signal_detected & ~GPS_activated)
    [detection_detected, speed, time, time_started] = actuador_signal(speed, YPred, distanceToBar, img, ...
        detection_detected, client, time, time_started);
end

```

*Figura 172. Bucle condicional para la detección de las señales*

Posteriormente, se añadirán 3 líneas de código a cada imagen devuelta, original y procesada, para informar al usuario de lo que está pasando en tiempo real de manera clara, a continuación se adjunta el código para la inserción de estas líneas y una captura ejemplificativa de la interfaz gráfica durante la ejecución.

```

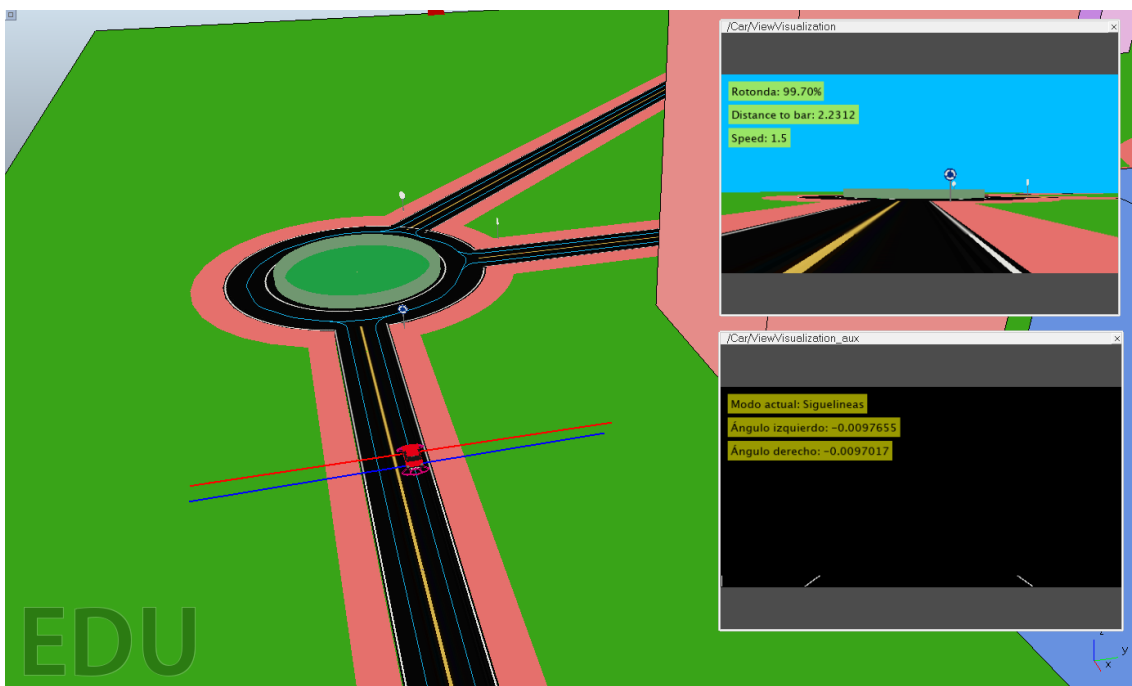
IMG_DISPLAY = insertText(IMG_DISPLAY, [10, 10], string(detection_text), 'TextColor', 'black', 'FontSize', 14);
IMG_DISPLAY = insertText(IMG_DISPLAY, [10, 40], ['Distance to bar: ' num2str(distanceToBar)], 'TextColor', 'black', 'FontSize', 14);
IMG_DISPLAY = insertText(IMG_DISPLAY, [10, 70], ['Speed: ' num2str(speed)], 'TextColor', 'black', 'FontSize', 14);

% Conversión de la imagen a RGB
bordes_carril_rgb = uint8(repmat(bordes_carril, [1, 1, 3]) * 255);

bordes_carril_rgb = insertText(bordes_carril_rgb, [10, 10], ['Modo actual: ' modo], 'TextColor', 'black', 'FontSize', 14);
bordes_carril_rgb = insertText(bordes_carril_rgb, [10, 40], ['Ángulo izquierdo: ' num2str(leftAngle)], 'TextColor', 'black', 'FontSize', 14);
bordes_carril_rgb = insertText(bordes_carril_rgb, [10, 70], ['Ángulo derecho: ' num2str(rightAngle)], 'TextColor', 'black', 'FontSize', 14);

```

**Figura 173. Implementación de la red neuronal y muestreo por pantalla**



**Figura 174. Imagen devuelta por el algoritmo de control**

En este punto, ya se han obtenido todos los parámetros necesarios para el control del vehículo y por lo tanto se procede con la transmisión de esta información desde Matlab a Coppelia. Para ello, se utilizará la función `setCarControl` que se ha mencionado en capítulos anteriores y que puede verse en la figura 156.

Además, se llamará a la función auxiliar `visualizaImagenCoppelia` para transmitir la imagen, con el texto añadido a Coppelia, esta función auxiliar junto con la implementación de esta etapa de transmisión de la información se adjunta a continuación.

```

setCarControl(sim, steer, lights, speed, leftAngle, rightAngle, lightOption);
visualizaImagenCoppelia(IMG_DISPLAY, sim, passiveVisionSensorHandle);
time = sim.getSimulationTime();

```

**Figura 175. Transmisión de la información de Matlab a Coppelia**

```
function visualizaImagenCoppelia(img, sim, handle)
    img = reshape(permute(flip(img, 1), [3 2 1]), [1 width(img) * height(img) * 3]);
    sim.setVisionSensorCharImage(handle, img);
end
```

*Figura 176. Función auxiliar visualizaImagenCoppelia*

Este algoritmo de control se ejecutará de manera ininterrumpida hasta que el vehículo llegue a su destino, para comprobar si esta condición se cumple, se calculará la distancia del GPS al destino y en caso de ser menor de 0.01 se detendrá la ejecución.

El cálculo de esta distancia se ha realizado de manera análoga a lo explicado en las figuras 139 y 140 y que se integra en este script mediante el siguiente código.

```
% Revisa si el vehículo ha completado el trayecto
x_diff = ListaPaths{length(ListaPaths)}.end_point{1} - gps{1};
y_diff = ListaPaths{length(ListaPaths)}.end_point{2} - gps{2};
distance = sqrt(x_diff^2 + y_diff^2);

% Se ha llegado al destino
if distance <= ACCEPTABLE_POINTS_DISTANCE
    disp('Se ha completado el trayecto')
    break;
end
```

*Figura 177. Comprobación para saber si se ha llegado al destino*

Finalmente una vez se ha llegado al destino, se ploteará la trayectoria realizada por el vehículo sobre la ideal para comprobar así el correcto desempeño del modelo a lo largo de toda la ejecución, una imagen ejemplificativa resultante de este proceso junto con el código que implementa esta funcionalidad se adjuntan a continuación.

```
% Graficar la trayectoria realizada
figure;
ruta_seleccionada(ListaPaths, gps);
hold on
plot(GPS_trayectoria_X, GPS_trayectoria_Y, 'b--', 'LineWidth', 1.5, 'DisplayName', 'Trayectoria real');
title('Trayecto completado');
xlabel('Posición X');
ylabel('Posición Y');
hold off
```

*Figura 178. Generación de la imagen resultante de la trayectoria realizada*



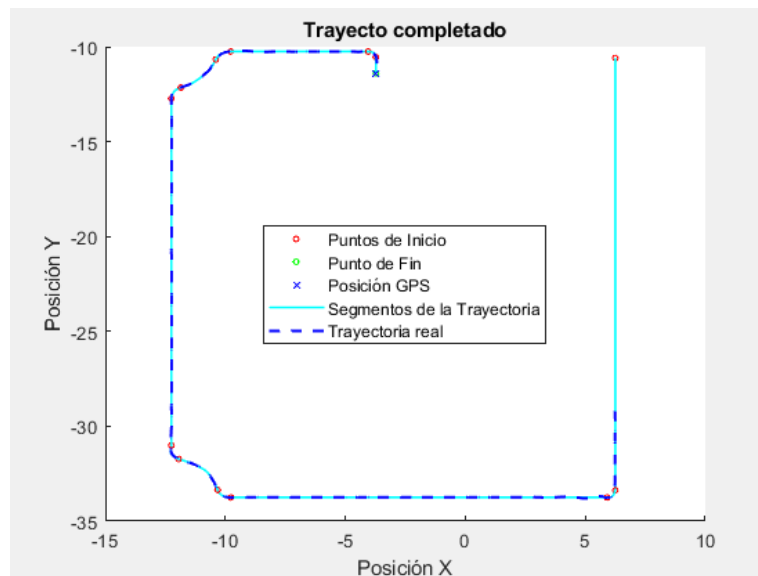


Figura 179. Imagen ejemplificativa tras completar el trayecto

Este script, integrará todas las técnicas utilizadas y trabajadas de manera conjunta y robusta permitiendo de igual manera diversas ampliaciones o modificaciones para versiones sucesivas o actualizaciones en el sistema en sí.

## 8. Estudio de aplicabilidad

Con este modelo, se ha logrado un sistema capaz de controlar un vehículo de manera completamente autónoma utilizando únicamente un sensor de visión y el gps del vehículo, además, se han implementado diferentes técnicas y estrategias de manera modular lo que permite que este modelo sea especialmente útil en entornos educativos o de investigación.

En los siguientes apartados, se desarrollan algunos campos o entornos donde este modelo se podría utilizar y algunas investigaciones o mejoras que este puede sufrir.

### 8.1. Aplicación del modelo

Este modelo puede ser muy útil en muy diversos campos ya que combina diferentes técnicas de visión por computador y de seguimiento de trayectorias de manera individual y conjunta. De esta forma, si el entorno de aplicación no es similar, se podrá adaptar fácilmente a cualquier necesidad que se tenga aunque debido a las diversas medidas de robustez que se han implementado no deberían ser necesarias demasiadas modificaciones.

Como se mencionó en la introducción de este capítulo, este modelo es un trabajo ideal para entornos educativos o de investigación ya que además de ofrecer una base muy sólida, y sobre todo un trabajo funcional y eficiente, se tiene una arquitectura modular donde cualquier pieza puede ser eliminada o modificada sin inhabilitar el modelo.

A continuación, se adjunta una lista con todos aquellos campos de investigación que han sido desarrollados a lo largo del diseño e implementación de este modelo.

- Modificación de un escenario de Coppelia para probar la conducción autónoma en situaciones de extrema dificultad.
- Análisis de las necesidades y dificultades que la conducción autónoma presenta y son necesarias en cualquier modelo de este campo.
- Comunicación funcional y en tiempo real entre Coppelia Sim y Matlab para optimizar las ventajas que cada una de estas ofrecen.
- Diseño e implementación de distintas redes neuronales para la detección de señales e identificación del entorno mediante la aplicación de Matlab Deep Network Designer.
- Aplicación de distintas técnicas de segmentación tanto en color (Color Thresholder), como binarias (recortes, bwlabel, regionprops, morfología matemática, etcétera).
- Aplicación del modelo pinhole para el cálculo de distancias a través de una sola cámara.
- Obtención y clasificación de la información obtenida desde Coppelia para su posterior trabajo en Matlab.
- Control de un vehículo con dirección Ackerman mediante la obtención de los ángulos de giro.
- Algoritmo para la obtención de la ruta más cercana mediante el uso de bucles anidados y cálculos de distancias en coordenadas cartesianas.
- Creación de una interfaz interactiva para que el usuario introduzca el destino al que el vehículo debe acudir.
- Obtención del grafo de conexiones de todas las rutas de un espacio delimitado y de la ruta más corta entre dos puntos utilizando este.
- Implementación de manera ordenada y sencilla de distintas técnicas de control en un mismo archivo independiente.
- Comprobación individual y colectiva de las partes del modelo ante todo tipo de circunstancias, tanto reales como extremas para comprobar el correcto funcionamiento de este.

## 8.2. Trabajos futuros

Como se ha ido comentando a lo largo de todo el trabajo, una de los principales objetivos y ventajas de este modelo era su escalabilidad y adaptabilidad por lo que las oportunidades para trabajos futuros serán muy diversas y extensas. A continuación, se enumeran algunas implementaciones que se podrían añadir a este proyecto aunque las oportunidades que este ofrece son prácticamente ilimitadas.

- Implementación de un control por interfaz gráfica en lugar de por texto.
- Creación de una aplicación móvil para el control de este modelo
- Implementación del modelo en un modelo real
- Adición de otro tipo de sensores tales como LIDARs, infrarrojos o ultrasonidos aunque en realidad prácticamente cualquier sensor es de utilidad en un campo tan complejo como la conducción autónoma
- Utilización de redes neuronales para todas las partes del proyecto
- Adición de situaciones nuevas al escenario tales como peatones, tráfico, etcétera.
- Nueva señalización y decisiones en relación con esta.
- Adaptación del modelo a entornos rurales o en los cuales no se puede observar con claridad el carril que debe seguir el vehículo.

- Detección de obstáculos en tiempo real y creación de una ruta alternativa en torno a este cambio de planes.

## 9. Conclusión

En este capítulo final se relacionarán aquellos objetivos planteados en el inicio del proyecto y las motivaciones por las que se llevó a cabo este modelo con el trabajo realizado a lo largo del proyecto. Aunque ha quedado patente a lo largo de los distintos capítulos, como estos objetivos se han ido cumpliendo en numerosas ocasiones, es importante recalcar la consecución de estos de manera individual.

El primero de ellos era el diseño e implementación del entorno de simulación lo que se ha cumplido de manera holgada y es que aunque la base del escenario donde tiene lugar la simulación sea la del repositorio de git, esta se ha modificado para añadir un tramo de curvas que pudiese al límite al vehículo. Además, se han creado escenarios extra tales como el utilizado para tomar las instantáneas de la base de datos, figura 40, o el utilizado para probar el control del vehículo en curvas, apartado 5.1.8.2.

El segundo, referido al conexionado entre las plataformas de Matlab y Coppelia, también se ha cumplido ya que en cada una de las partes del algoritmo de control se ha intercambiado información entre ambas plataformas de manera reiterada y eficiente.

El tercero objetivo que se tenía en este proyecto era la creación de una red neuronal para analizar el entorno y actuar en consecuencia, como se desarrolló a lo largo del capítulo 4, se han creado 3 redes neuronales de las cuales dos de ellas cumplen completamente con este objetivo. Además, se ha creado una función auxiliar para el control del vehículo en función de las señales que han sido detectadas cumpliendo así con la segunda parte de este.

El objetivo relacionado con los algoritmos de visión por computación se ha cumplido mediante las técnicas provistas en el capítulo 5 para la segmentación de la imagen en la umbralización del carril y la barra de las señales. Esta parte del modelo es una de las partes más importantes del sistema y por lo tanto se puede concluir que este objetivo se logra satisfacer de manera excelsa.

Respecto al penúltimo objetivo, la integración del sistema de GPS y planificación de rutas, aunque esta sea la única parte del algoritmo que no se ha realizado mediante técnicas de visión también se ha cumplido completamente. Más concretamente, la parte del proyecto que cumple este objetivo es el capítulo 6, donde se desarrollan las técnicas que cumplen este.

El último objetivo, y el más importante, es la creación de un modelo modular y escalable, objetivo, sobre el que se ha basado todo el proyecto y que ha quedado sobradamente cumplido como se puede observar en el punto 8.2, donde queda claro que las ampliaciones, aunque no son imprescindibles, son numerosas y diversas.

En conclusión, se puede afirmar que el proyecto ha cumplido con los objetivos y motivaciones que llevaron a la realización de este con grandes resultados en todo tipo de situaciones. Mediante

este trabajo, se ha logrado poner en valor la complejidad de un campo tan novedoso como la conducción autónoma y las posibilidades que este ofrece para la combinación de manera eficiente de distintas técnicas y algoritmos.

## Bibliografía

---

- [1] «CFP | UPV | APLICACIÓN DE LA ROBÓTICA EDUCATIVA EN EL AULA». *Centro de Formación Permanente | UPV*, <https://www.cfp.upv.es>.
- [2] Fiat, Equipe Automax. «Sensor de estacionamiento: entenda como funciona a tecnologia». *Automax Fiat*, 7 de agosto de 2023, <https://www.automaxfiat.com.br/sensor-de-estacionamento/>.
- [3] López, José María. «El vehículo autónomo es el futuro, pero tiene más años que la mayoría de nosotros». *Hipertextual*, 15 de agosto de 2020, <http://hipertextual.com/2020/08/origen-historia-vehiculo-autonomo>.
- [4] López, José María. «Un paseo por Futurama: ¿cómo imaginaron el futuro en 1939?». *Hipertextual*, 26 de junio de 2021, <http://hipertextual.com/2021/06/futurama-feria-mundial-futuro>.
- [5] interior, DGT, Ministerio. *Sistemas avanzados de ayuda a la conducción (ADAS)*. <https://www.dgt.es/muevete-con-seguridad/conviertete-en-un-buen-conductor/Sistemas-avanzados-de-ayuda-a-la-conduccion-ADAS-/>.
- [6] *Accidentes de tráfico, en datos y estadísticas*. <https://www.epdata.es/datos/accidentes-traffic-datos-estadisticas/65/espana/106>.
- [7] Réyez, José. «El 95 por ciento de accidentes vehiculares, por factor humano: Inegi». *Contralínea*, 28 de noviembre de 2021, <https://contralinea.com.mx/interno/semana/el-95-por-ciento-de-accidentes-vehiculares-por-factor-humano-inegi/>.
- [8] *SAE Levels of Driving Automation<sup>TM</sup> Refined for Clarity and International Audience*. <https://www.sae.org/site/blog/sae-j3016-update>.

- [9] «The Man Who Invented the Self-Driving Car (in 1986)». *POLITICO*, 19 de julio de 2018, <https://www.politico.eu/article/delf-driving-car-born-1986-ernst-dickmanns-mercedes/>.
- [10] «European Eureka Project - PROgraMme for a European Traffic of Highest Efficiency and Unprecedented Safety (PROMETHEUS)». *University of Portsmouth*, <https://researchportal.port.ac.uk/en/projects/european-eureka-project-programme-for-a-european-traffic-of-highe>.
- [11] «Waymo - Vehículos autónomos - Automóviles que se conducen a sí mismos - Transporte privado a pedido». *Waymo*, <https://waymo.com/intl/es/>.
- [12] «Self-Driving Giant Waymo on the Verge of Bringing Robotaxis to New U.S. City». *The US Sun*, 19 de octubre de 2022, <https://www.the-sun.com/motors/6481674/self-driving-waymo-los-angeles/>.
- [13] Torre, Alberto de la. «La primera F1 de monoplasas autónomos acaba de desvelar su primer coche: a 300 km/h sin conductor». *Xataka*, 16 de octubre de 2023, <https://www.xataka.com/movilidad/primera-f1-monoplazas-autonomos-acaba-desvelar-su-primer-coche-a-300-km-h-conductor>.
- [14] Juanatey Hermo, Daniel Andrés. *Creación de un modelo de simulación de robot móvil para conducción autónoma*. 2020. *ruc.udc.es*, <https://ruc.udc.es/dspace/handle/2183/30488>.
- [15] Ladero García, Rodrigo. *Detección de señales y líneas de carril para conducción autónoma con vehículo a escala*. febrero de 2019, <https://oa.upm.es/54102/>.
- [16] Collado Hernáiz, Juan Manuel. *Detección y modelado de carriles de vías interurbanas mediante análisis de imágenes para un sistema de ayuda a la conducción*. 2009. Universidad Carlos III de Madrid, <http://purl.org/dc/dcmitype/Text>. *dialnet.unirioja.es*, <https://dialnet.unirioja.es/servlet/tesis?codigo=20284>

- [17] Díaz Llorca, Daniel. *Simulación de un entorno industrial mediante la herramienta de trabajo CoppeliaSim (V-Rep)*. 2020. Universitat Politècnica de València, Proyecto/Trabajo fin de carrera/grado. [riunet.upv.es](http://riunet.upv.es), <https://riunet.upv.es/handle/10251/153389>.
- [18] Saiz Allende, Alejandro. *Implementación de redes neuronales para conducción autónoma*. marzo de 2021. [repositorio.unican.es](http://repositorio.unican.es), <https://repositorio.unican.es/xmlui/handle/10902/21899>.
- [19] <https://github.com/klima7/Autonomous-car-simulation>
- [20] Llopis Sánchez, Andreu. *Sistema Inteligente para la planificación y guiado de vehículos*. 2023. Universitat Politècnica de València, Proyecto/Trabajo fin de carrera/grado. [riunet.upv.es](http://riunet.upv.es), <https://riunet.upv.es/handle/10251/199834>.
- [21] «El origen de los planificadores de rutas». *Cómo llegar*, <https://comollegar.app/origen/>.
- [22] «La línea del tiempo del GPS: Desde su origen hasta el presente. Línea de Tiempo». *Línea de Tiempo*, 13 de enero de 2023, <https://lineadetiempo.net/la-linea-del-tiempo-del-gps-desde-su-origen-hasta-el-presente/>.
- [23] «Diseño y planificación optimizada de rutas en todo el mundo - Google Maps Platform». *Google Maps Platform*, <https://mapsplatform.google.com/intl/es-419/maps-products/routes/>.
- [24] «Algoritmo de la ruta más corta de Dijkstra - Introducción gráfica y detallada». *freeCodeCamp.org*, 24 de octubre de 2022, <https://www.freecodecamp.org/espanol/news/algoritmo-de-la-ruta-mas-corta-de-dijkstra-introduccion-grafica/>.
- [25] «Robotic Path Planning». *Path Planning*, [https://fab.cba.mit.edu/classes/865.21/topics/path\\_planning/robotic.html](https://fab.cba.mit.edu/classes/865.21/topics/path_planning/robotic.html).

- [26] Digiltea. «Los algoritmos que hacen posible la conducción autónoma». *Digiltea, tu aliado digital.*, 30 de marzo de 2022, <https://digiltea.com/los-algoritmos-que-hacen-posible-la-conduccion-autonoma/>.
- [27] Pek, Christian, et al. «Using Online Verification to Prevent Autonomous Vehicles from Causing Accidents». *Nature Machine Intelligence*, vol. 2, n.º 9, septiembre de 2020, pp. 518-28. *DOI.org (Crossref)*, <https://doi.org/10.1038/s42256-020-0225-y>.
- [28] Redes neuronales para conducción autónoma,  
<https://repositorio2.unican.es/xmlui/bitstream/handle/10902/21899/434502.pdf?sequence=1>.
- [29] Álvarez, Raúl. «El director de inteligencia artificial de Tesla nos explica en vídeo cómo entrenan sus redes neuronales para el uso de Autopilot». *Xataka*, 12 de noviembre de 2019, <https://www.xataka.com/robotica-e-ia/director-inteligencia-artificial-tesla-nos-explica-video-como-entrenan-sus-redes-neuronales-para-uso-autopilot>.
- [30] *Robot simulator CoppeliaSim: create, compose, simulate, any robot - Coppelia Robotics*. <https://www.coppeliarobotics.com/>.
- [31] *BubbleRob tutorial*.  
<https://manual.coppeliarobotics.com/en/externalControllerTutorial.htm>.
- [32] *CoppeliaSim User Manual*. <https://manual.coppeliarobotics.com/>.
- [33] *CoppeliaSim*.  
[https://es.mathworks.com/products/connections/product\\_detail/coppelasim.html](https://es.mathworks.com/products/connections/product_detail/coppelasim.html).
- [34] *MATLAB para estudiantes*. <https://es.mathworks.com/products/matlab/student.html>.  
Accedido 24 de abril de 2024.
- [35] *Deep Learning Toolbox*. <https://es.mathworks.com/products/deep-learning.html>.  
Accedido 24 de abril de 2024.



- [36] *Diseñar, visualizar y entrenar redes de deep learning - MATLAB - MathWorks España.*  
<https://es.mathworks.com/help/deeplearning/ref/deepnetworkdesigner-app.html>.
- [37] Krizhevsky, Alex, et al. «ImageNet Classification with Deep Convolutional Neural Networks». *Communications of the ACM*, vol. 60, n.º 6, mayo de 2017, pp. 84-90.  
*DOI.org (Crossref)*, <https://doi.org/10.1145/3065386>.
- [38] *Image Processing Toolbox.* <https://es.mathworks.com/products/image-processing.html>.
- [39] *Umbrales de imágenes en color - MATLAB - MathWorks España.*  
<https://es.mathworks.com/help/images/ref/colorthresholder-app.html>.
- [40] Calvo, Diego. «Red Neuronal Convolutiva CNN». *Diego Calvo*, 20 de julio de 2017,  
<https://www.diegocalvo.es/red-neuronal-convolutiva/>.
- [41] *¿Qué Son Las Redes Neuronales Convolutivas? (Parte 2) – Innovation through Artificial Intelligence.* <https://pixelabs.es/que-son-las-redes-neuronales-convolutivas-parte-2/>.
- [42] Ramadhan, Awf A., y Muhammet Baykara. «A Novel Approach to Detect COVID-19: Enhanced Deep Learning Models with Convolutional Neural Networks». *Applied Sciences*, vol. 12, n.º 18, enero de 2022, p. 9325. *www.mdpi.com*,  
<https://doi.org/10.3390/app12189325>.
- [43] *Tipos de operaciones morfológicas - MATLAB & Simulink - MathWorks España.*  
<https://es.mathworks.com/help/images/morphological-dilation-and-erosion.html>.
- [44] Redmon, Joseph, y Ali Farhadi. «YOLO9000: Better, Faster, Stronger». *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2017, pp. 6517-25. *DOI.org (Crossref)*, <https://doi.org/10.1109/CVPR.2017.690>.
- [45] *Detect vehicles using YOLO v2 Network - MATLAB vehicleDetectorYOLOv2 - MathWorks España.*  
<https://es.mathworks.com/help/driving/ref/vehicledetector yolov2.html>.

- [46] *YOLO: Algorithm for Object Detection Explained*.  
<https://www.linkedin.com/pulse/yolo-algorithm-object-detection-explained-arnab-mukherjee>.
- [47] Fernandez, Nekane & Moran, Adrian & Amurrio, Andoni. (2023). *Optimizing Distributed Artificial Intelligence in Edge-To-Cloud Continuum scenarios*.  
10.13140/RG.2.2.15711.64164.
- [48] Hargurjeet. «7 Best Techniques To Improve The Accuracy of CNN W/O Overfitting». *Medium*, 27 de mayo de 2021, <https://gurjeet333.medium.com/7-best-techniques-to-improve-the-accuracy-of-cnn-w-o-overfitting-6db06467182f>.
- [49] «Overfitting. Qué es, causas, consecuencias y cómo solucionarlo». *Grupo Atico34*, 20 de febrero de 2021, <https://protecciondatos-lopd.com/empresas/overfitting/>.
- [50] *Clasificar observaciones mediante análisis discriminante - MATLAB classify - MathWorks España*. <https://es.mathworks.com/help/stats/classify.html>.
- [51] *Medir propiedades de regiones de imágenes - MATLAB regionprops - MathWorks España*. <https://es.mathworks.com/help/images/ref/regionprops.html>.
- [52] *Eliminar objetos pequeños de una imagen binaria - MATLAB bwareaopen - MathWorks España*. [https://es.mathworks.com/help/images/ref/bwareaopen\\_es.html](https://es.mathworks.com/help/images/ref/bwareaopen_es.html).
- [53] *Encontrar los bordes de una imagen 2D en escala de grises - MATLAB edge - MathWorks España*. <https://es.mathworks.com/help/images/ref/edge.html>.
- [54] Suárez, Patricia, y Mónica Villavicencio. «Detección de Contornos utilizando el Algoritmo Canny en Imágenes Cross-Espectrales Fusionadas». *Enfoque UTE*, vol. 8, n.º 1, febrero de 2017, pp. 16-30. [www.redalyc.org](http://www.redalyc.org),  
<https://www.redalyc.org/journal/5722/572262176002/html/>.
- [55] Pera, Publicado por Sr. *Formula SAE: La Dirección (Ackermann)*.  
<http://www.fuelwasters.com/2011/04/formula-sae-la-direccion-ackermann.html>.

- [56] Kumar, Prabu. «How to Choose the Right Lens for Your Embedded Camera Application». *E-Con Systems*, 22 de julio de 2022, <https://www.e-consystems.com/blog/camera/technology/how-to-choose-the-right-lens-for-your-embedded-camera-application/>.
- [57] *Calcular La Distancia Entre Dos Puntos En El Plano Cartesiano*. 25 de septiembre de 2021, <https://trilosangulo.blogspot.com/2021/09/calcular-la-distancia-entre-dos-puntos.html>.
- [58] *Shortest path between two single nodes - MATLAB shortestpath - MathWorks España*. <https://es.mathworks.com/help/matlab/ref/graph.shortestpath.html>.



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



# UNIVERSITAT POLITÈCNICA DE VALÈNCIA

**Escuela Técnica Superior de Ingeniería del Diseño**

---

## **SISTEMA DE VISIÓN INTELIGENTE PARA EL GUIADO DE UN VEHÍCULO AUTÓNOMO EN UN ENTORNO URBANO**

**TRABAJO FINAL DE GRADO**

**GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL  
Y AUTOMÁTICA**

**Documento**

**Anexos**

## Anexos

---

|   |     |
|---|-----|
| 1. Códigos completos .....  | 140 |
| 1.1. Código captura imágenes para red neuronal .....                    | 140 |
| 1.2. Código completo resizing.....                                      | 141 |
| 1.3. Código completo para el Split de las imágenes.....                 | 142 |
| 1.4. Código completo test CNN .....                                     | 143 |
| 1.5. Código completo video test CNN .....                               | 145 |
| 1.6. Código para el entrenamiento de la red YOLO v2.....                | 147 |
| 1.7. Código de la función para la detección de señales.....             | 148 |
| 1.8. Código para la actuación ante las señales .....                    | 149 |
| 1.9. Código de la función para la detección del carril.....             | 150 |
| 1.10. Código completo para las pruebas de la detección del carril. .... | 152 |
| 1.11. Código de la detección de la barra .....                          | 154 |
| 1.12. Código completo para mostrar la ruta óptima .....                 | 156 |
| 1.13. Código completo para el seguimiento del GPS.....                  | 157 |
| 1.14. Código completo del script para las pruebas de GPS.....           | 158 |
| 1.15. Código completo del script principal de integración.....          | 160 |
| 2. Agenda 2030, ODS.....  | 163 |

## 1. Códigos completos

### 1.1. Código captura imágenes para red neuronal

#### Creación de la base de datos

```
clearvars;
close all;

% Inicio simulación
client = RemoteAPIClient();
sim = client.getObject('sim');
fprintf('Program started\n');
client.setStepping(true);
sim.startSimulation();

client.step();

startTime = sim.getSimulationTime();
t = startTime;

% Ángulos iniciales sobre los que aplicar variación (en grados)
angulos_iniciales = [86.389, 0.01, -0.383];

% Número de imágenes a capturar
inicio_img = 0;
num_imagenes = inicio_img + 100;

% Rango de variación de orientación de la cámara
rango_orientacion = 5;

% Rango de variación de posición en los ejes x e y
rango_posicion_x = 0.3;
rango_posicion_y = 0.3;

% Contador de imágenes capturadas
capturas_realizadas = inicio_img;

while t - startTime < 20 && capturas_realizadas < num_imagenes
    % Generar y aplicar orientaciones y posiciones aleatorias
    for i = inicio_img:num_imagenes
        client.step();

        visionSensorHandle = sim.getObject('/ViewCamera');

        % Capturar imagen
        img = getImage(sim, visionSensorHandle);

        % Generar ángulos aleatorios para los ejes X, Y y Z
        angulos_aleatorios = randi([-rango_orientacion, rango_orientacion], 1, 3);

        % Valor entre -rango_posicion_x y rango_posicion_x
        posicion_x_aleatoria = (2*rand - 1) * rango_posicion_x;
        % Valor entre -rango_posicion_y y rango_posicion_y
        posicion_y_aleatoria = (2*rand - 1) * rango_posicion_y;

        % Aplicar variación a los ángulos iniciales
        angulos_nuevos = angulos_iniciales + angulos_aleatorios;

        % Enviar comando para establecer la orientación del sensor
        sim.setObjectOrientation(visionSensorHandle, -1, angulos_nuevos*pi/180);

        % Obtener la posición actual del sensor
        posicion_actual = sim.getObjectPosition(visionSensorHandle, -1);
```

## Creación de la base de datos

```
% Acceder al contenido de la celda para obtener el vector numérico
posicion_actual = cell2mat(posicion_actual);

% Aplicar variación a la posición en los ejes X e Y
nueva_posicion_x = posicion_actual(1) + posicion_x_aleatoria;
nueva_posicion_y = posicion_actual(2) + posicion_y_aleatoria;

% Establecer la nueva posición del sensor
nueva_posicion = [nueva_posicion_x, nueva_posicion_y, posicion_actual(3)];
sim.setObjectPosition(visionSensorHandle, -1, nueva_posicion);

% Reorganizar y guardar la imagen en formato PNG
nombre_archivo = fullfile('.../.../Imagenes\Stop', sprintf('Stop_%d.png',
capturas_realizadas + 1));
imwrite(img, nombre_archivo);

disp(['Imagen ', num2str(capturas_realizadas + 1), ' capturada y guardada']);

capturas_realizadas = capturas_realizadas + 1;
end
t = sim.getSimulationTime();
end

sim.stopSimulation();

fprintf('Program ended\n');

function img = getImage(sim, handle)
[img, resX, resY] = sim.getVisionSensorCharImage(handle);
img = flip(permute(reshape(img, 3, resX, resY), [3 2 1]), 1);
end
```

## 1.2. Código completo resizing

### Redimensionado de las imágenes de la base de datos

```
%% Resizing the images for a correct labelling
clear all
clc
folderName = '../.../Imagenes\Images Sin entreno';

% Se recorren todas las imágenes de la carpeta 'folderName'
srcDir = fullfile(folderName);
srcFiles = dir(fullfile(srcDir, '**', '*.png'));

imageResizingNeeded = true;
if imageResizingNeeded
    for i = 1 : length(srcFiles)
        filename = fullfile(srcFiles(i).folder, srcFiles(i).name);
        % Comprobar si es un directorio
        if ~isfile(filename)
            continue; % Saltar directorios
        end
        im = imread(filename);

        % Redimensionar las imágenes
        k = imresize(im, [224, 224]);

        % Obtener la ruta relativa
        relPath = extractAfter(filename, folderName);

        % Creación del nombre del archivo
        [~, name, ~] = fileparts(srcFiles(i).name);
        newfilename = fullfile(srcFiles(i).folder, 'Images_resized', [name, '_resized.png']);

        % Crear la carpeta destino si esta no existe
        if ~isfolder(fullfile(srcFiles(i).folder, 'Images_resized'))
```

### Redimensionado de las imágenes de la base de datos

```
mkdir(fullfile(srcFiles(i).folder, 'Images_resized'));  
end  
  
% Escribir imagen redimensionada  
imwrite(k, newfilename, 'png');  
end  
end
```

## 1.3. Código completo para el Split de las imágenes

### División de las imágenes de la base de datos

```
%% Splitting the images for training and validation  
clear all  
clc  
  
folderName = '../../Images resized/Images sin edificios';  
srcDir = fullfile(folderName);  
srcSubfolders = dir(fullfile(srcDir, '**'));  
  
% Eliminar carpetas que no sean subcarpetas  
srcSubfolders = srcSubfolders([srcSubfolders.isdir] & ~strcmp({srcSubfolders.name}, '.') &  
~strcmp({srcSubfolders.name}, '..'));  
  
% Proporciones de división  
trainRatio = 0.7;  
valRatio = 0.2;  
testRatio = 0.1;  
  
% Iterar sobre las subcarpetas  
for i = 1:length(srcSubfolders)  
    subfolderName = srcSubfolders(i).name;  
    subfolderPath = fullfile(srcDir, subfolderName);  
  
    % Obtener las imágenes dentro de la subcarpeta  
    srcFiles = dir(fullfile(subfolderPath, '*.png'));  
    numImages = length(srcFiles);  
  
    % Obtener los índices de división  
    [numTrain, numVal, numTest] = dividerand(numImages, trainRatio, valRatio, testRatio);  
  
    % Crear las carpetas para cada conjunto dentro de la subcarpeta  
    trainFolder = fullfile(folderName, 'Train/', subfolderName);  
    valFolder = fullfile(folderName, 'Validation/', subfolderName);  
    testFolder = fullfile(folderName, 'Test/', subfolderName);  
  
    if ~isfolder(trainFolder)  
        mkdir(trainFolder);  
    end  
  
    if ~isfolder(valFolder)  
        mkdir(valFolder);  
    end  
  
    if ~isfolder(testFolder)  
        mkdir(testFolder);  
    end  
  
    imgName = srcFiles(i).name;  
    imgPath = fullfile(subfolderPath, imgName);  
    k = 0;  
    % Copiar las imágenes a las carpetas correspondientes  
    for k = 1:length(numTrain)  
        imgName = srcFiles(k).name;  
        imgPath = fullfile(subfolderPath, imgName);  
        copyfile(imgPath, fullfile(trainFolder, imgName));  
    end  
end
```



### División de las imágenes de la base de datos

```
for k = (length(numTrain)+1):(length(numTrain) + length(numVal))
    imgName = srcFiles(k).name;
    imgPath = fullfile(subfolderPath, imgName);
    copyfile(imgPath, fullfile(valFolder, imgName));
end

for k = (length(numTrain) + length(numVal)+1):numImages
    imgName = srcFiles(k).name;
    imgPath = fullfile(subfolderPath, imgName);
    copyfile(imgPath, fullfile(testFolder, imgName));
end
end

disp('Imágenes divididas y almacenadas en carpetas Train, Validation y Test.');
```

## 1.4. Código completo test CNN

### Testeo de las imágenes para las CNN

```
% Testeo de una CNN
close all
clear all
clc
% Cargar el modelo que se desee comprobar
red = load('SqueezeNet_mvaltor.mat');

folderName = '.././Images resized/Images resized 227x227/Test';
srcDir = fullfile(folderName);
srcSubfolders = dir(fullfile(srcDir, '*'));

certeza_acierto = [];
certeza_fallo = [];
img_fallo = {};

% Contador de imagen para el eje x
img_count_aciertos = 1;
img_count_fallos = 1;
count_aciertos = 0;
count_fallos = 0;

% Crear una carpeta para las detecciones fallidas
outputFolder = fullfile(srcDir, '../detecciones_fallidas');
if ~isfolder(outputFolder)
    mkdir(outputFolder);
end

% Crear una figura para mostrar las probabilidades de detección de aciertos y fallos
figure;

% Iterar sobre las subcarpetas
for i = 1:length(srcSubfolders)
    subfolderName = srcSubfolders(i).name;

    % Ignorar las carpetas '.' y '..'
    if strcmp(subfolderName, '.') || strcmp(subfolderName, '..')
        continue;
    end

    subfolderPath = fullfile(srcDir, subfolderName);

    % Obtener las imágenes dentro de la subcarpeta
    srcFiles = dir(fullfile(subfolderPath, '*.png'));

    % Iterar sobre las imágenes en la subcarpeta
    for j = 1:length(srcFiles)
        % Leer la imagen
```

## Testeo de las imágenes para las CNN

```
imgName = srcFiles(j).name;
imgPath = fullfile(subfolderPath, imgName);
I = imread(imgPath);

% Obtener el nombre de la subcarpeta como label
true_label = subfolderName;

%Clasificar la imagen de prueba usando la red neuronal entrenada
[YPred, probs] = classify(red.Red_SqueezeNet_mvaltor, I);

% Almacenar las probabilidades de detección
certeza = max(probs);

% Obtener el color para la clase detectada
color = seleccion_colores(YPred);

% Verificar si la detección es correcta
if YPred == true_label
    count_aciertos = count_aciertos + 1;
    certeza_acierto = [certeza_acierto; certeza];
    % Mostrar la probabilidad de detección en la figura de aciertos
    subplot(3, 1, 1);
    scatter(img_count_aciertos, 100*certeza, 7, color, 'filled');
    hold on;
    img_count_aciertos = img_count_aciertos + 1;
else
    count_fallos = count_fallos + 1;
    certeza_fallo = [certeza_fallo; certeza];
    % Mostrar la probabilidad de detección en la figura de fallos
    subplot(3, 1, 2);
    scatter(img_count_fallos, 100*certeza, 7, color, 'filled');
    hold on;
    img_fallo{end+1} = imgPath;
    img_count_fallos = img_count_fallos + 1;
    % Crear un nuevo nombre de archivo para la imagen fallida
    [~, imgNameWithoutExt, ext] = fileparts(imgName);
    newImgName = ['SqueezeNet_mvaltor_', subfolderName, '_', YPred,
num2str(count_fallos), ext];
    % Guardar el nombre de la imagen fallida en el vector img_fallo
    img_fallo{end+1} = newImgName;
    % Copiar la imagen fallida a la carpeta "detecciones_fallidas"
    copyfile(imgPath, fullfile(outputFolder, strjoin(string(newImgName), '')));
    img_count_fallos = img_count_fallos + 1;
end
end
end

accuracy = count_aciertos / (count_aciertos + count_fallos) * 100;

% Configurar ejes y etiquetas para la figura de aciertos
subplot(3, 1, 1);
legendLabels = {'Callejon', 'Limite velocidad 30', 'Parking', 'Paso de peatones', 'Prohibido',
'Reversed', 'Rotonda', 'Semáforo', 'Sentido unico', 'Stop'};
legendColors = {'r', 'g', 'b', 'c', 'm', 'y', 'k', [0.5 0.5 0.5], [1 0.5 0], [0 1 1]};
legendEntries = containers.Map(legendLabels, legendColors);
legendEntriesKeys = keys(legendEntries);
legendEntriesPlot = cell(1, numel(legendEntriesKeys));

for i = 1:numel(legendEntriesKeys)
    hold on;
    legendEntriesPlot{i} = scatter(nan, nan, 70, legendColors{i}, 'filled', 'DisplayName',
legendEntriesKeys{i});
end

legend([legendEntriesPlot{:}], legendEntriesKeys, 'Location', 'best');

titleTextAciertos = "Probabilidad Media Aciertos: " + num2str(mean(certeza_acierto), 3);
xlabel('Número de imágenes');
ylabel('Confianza');
```

## Testeo de las imágenes para las CNN

```
title(titleTextAciertos);
colorbar;
xlim([0 img_count_aciertos]);
ylim([60 100]);
colorbar('off');
grid minor

% Configurar ejes y etiquetas para la figura de fallos
subplot(3, 1, 2);
for i = 1:numel(legendEntriesKeys)
    hold on;
    scatter(nan, nan, 70, legendColors{i}, 'filled', 'DisplayName', legendEntriesKeys{i});
end
legend([legendEntriesPlot{:}], legendEntriesKeys, 'Location', 'best');
titleTextFallos = "Probabilidad Media Fallos: " + num2str(mean(certeza_fallo), 3);
xlabel('Número de imágenes');
ylabel('Confianza');
title(titleTextFallos);
colorbar;
xlim([0 img_count_fallos]);
ylim([60 100]);
colorbar('off');
grid minor

% Añadir subplot extra solo para la leyenda
subplot(3, 1, 3);
axis off; % Ocultar los ejes en este subplot

% Crear la leyenda para ambos subplots
for i = 1:numel(legendEntriesKeys)
    hold on;
    scatter(nan, nan, 70, legendColors{i}, 'filled', 'DisplayName', legendEntriesKeys{i});
end

legend([legendEntriesPlot{:}], legendEntriesKeys, 'Location', 'best');

% Mostrar resultados
disp('Resultados:');
disp(['Número de detecciones correctas:', num2str(count_aciertos)]);
disp(['Número de errores de detección:', num2str(count_fallos)]);
disp(['Porcentaje de acierto:', num2str(accuracy), '%']);
disp(['Certidumbre en detecciones correctas:', num2str(mean(certeza_acierto))]);
disp(['Certidumbre en errores de detección:', num2str(mean(certeza_fallo))]);
disp('Figuras con las probabilidades de detección y el valor medio generadas.');
```

## 1.5. Código completo video test CNN

### Creación del video del testeo de las CNN

```
%% Comprobación Test Images
clear all
clc
% Cargar el modelo que se desee comprobar
% load('SqueezeNet_mvalor\Red_SqueezeNet_mvalor.mat')
% red = load('YOLOv2/modelo2/YOLO_modelo2.mat');
red = load('Resnet50_mvalor.mat');
folderName = '../Images resized/Images sin edificios\Test';
% folderName = '../Images resized/Images resized 227x227/Test/';
% folderName = 'YOLOv2/modelo2/Test';

srcDir = fullfile(folderName);
srcSubfolders = dir(fullfile(srcDir, '*'));

det_probs = [];
% Contador de imagen para el eje x
```

## Creación del video del testeo de las CNN

```
img_count = 1;

% Crear el video de salida
outputVideo = VideoWriter('test_Resnet50_mvaltor.avi');
open(outputVideo);

% Iterar sobre las subcarpetas
for i = 1:length(srcSubfolders)
    subfolderName = srcSubfolders(i).name;
    subfolderPath = fullfile(srcDir, subfolderName);

    % Obtener las imágenes dentro de la subcarpeta
    srcFiles = dir(fullfile(subfolderPath, '*.png'));

    % Iterar sobre las imágenes en la subcarpeta
    for j = 1:length(srcFiles)
        % Leer la imagen
        imgName = srcFiles(j).name;
        imgPath = fullfile(subfolderPath, imgName);
        I = imread(imgPath);

        % Clasificar la imagen de prueba usando la red neuronal entrenada
        [YPred,probs] = classify(red.Resnet50_mvaltor, I);
        label = YPred;

        % Mostrar el título en el video
        titleText = string(label) + ": " + num2str(100*max(probs),3) + "%";
        I = insertText(I, [10 10], titleText, 'FontSize', 16, 'BoxColor', 'green',
            'BoxOpacity', 0.3, 'TextColor', 'white');

        % Mostrar el frame en el video durante medio segundo (30 frames por segundo)
        for k = 1:15 % 30 frames por segundo * 0.5 segundos = 15 frames
            writeVideo(outputVideo, I);
        end

        % Mostrar la imagen con el título
        imshow(I);
        drawnow;

        % Esperar 0.2 segundos antes de pasar a la siguiente imagen
        pause(0.1);

        % Almacenar las probabilidades de detección
        det_probs = [det_probs; max(probs)];

        % Obtener el color para la clase detectada
        color = seleccion_colores(YPred);

        % Mostrar la probabilidad de detección en la figura
        scatter(img_count, 100*max(probs), 7, color, 'filled');
        hold on;

        % Actualizar el título con el valor medio de las probabilidades
        mean_prob = mean(det_probs);

        % Incrementar el contador de imagen
        img_count = img_count + 1;
    end
end

close(outputVideo);
```

## 1.6. Código para el entrenamiento de la red YOLO v2

### Entrenamiento de la red propia YOLO v2

```
%% YOLO design con aumento de data base
clear vars;
close all;
clc

% Ruta a la carpeta que contiene las subcarpetas para cada clase
folderName_train = 'modelo2/Train/';
srcDir_train = fullfile(folderName_train);

folderName_validation = 'modelo2/Validation/';
srcDir_validation = fullfile(folderName_validation);

folderName_test = 'modelo2/Test/';
srcDir_test = fullfile(folderName_test);

% Carga los datos
imds_train = imageDatastore(srcDir_train, 'IncludeSubfolders', true, 'LabelSource',
'foldernames');
imds_validation = imageDatastore(srcDir_validation, 'IncludeSubfolders', true, 'LabelSource',
'foldernames');
imds_test = imageDatastore(srcDir_test, 'IncludeSubfolders', true, 'LabelSource',
'foldernames');

%Se someten las imágenes a rotaciones y transformaciones para aumentar el
%dataset
%Input size para YOLOv2
input_size = [416 416 3];

% Crear el augmentedImageDatastore con transformaciones
augmentedDS_train = augmentedImageDatastore(input_size, imds_train);
augmentedDS_validation = augmentedImageDatastore(input_size, imds_validation);
augmentedDS_test = augmentedImageDatastore(input_size, imds_test);

% Número total de imágenes de cada tipo
trainingIdx = numel(imds_train.Files);
validationIdx = numel(imds_validation.Files);
testIdx = numel(imds_test.Files);

% Conjunto de datos de entrenamiento
trainingData = augmentedDS_train.subset(1:trainingIdx);

% Conjunto de datos de validación
validationData = augmentedDS_validation.subset(1:validationIdx);

% Conjunto de datos de prueba
testData = augmentedDS_test.subset(1:testIdx);

% Red Propia
numClasses = numel(categories(imds_train.Labels));
layers = [
    imageInputLayer(input_size)

    convolution2dLayer(3,8, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, 'Stride', 2)

    convolution2dLayer(3,16, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, 'Stride', 2)

    convolution2dLayer(3,32, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(numClasses)
```

## Entrenamiento de la red propia YOLO v2

```
softmaxLayer
classificationLayer];

% Agrega regularización L2
weight_decay = 0.001;
layers(2).Weights = layers(2).Weights / (1 + weight_decay);
layers(6).Weights = layers(6).Weights / (1 + weight_decay);
layers(10).Weights = layers(10).Weights / (1 + weight_decay);

% Agrega capas de dropout
layers(4) = dropoutLayer(0.3);
layers(8) = dropoutLayer(0.3);

options = trainingOptions('sgdm', ...
    'MiniBatchSize', 10, ...
    'MaxEpochs', 25, ...
    'ValidationData', validationData, ...
    'ValidationFrequency', 150, ...
    'InitialLearnRate', 1e-4, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropPeriod', 20, ...
    'LearnRateDropFactor', 0.2, ...
    'Plots', 'training-progress');

% Entrenar la red
YOLO_modelo2 = trainNetwork(trainingData, layers, options);
```

## 1.7. Código de la función para la detección de señales

### Función deteccin\_signal\_mvalor

```
function [debounce, detection_text, YPred, YPred_ant, signal_detected] =
deteccion_signal_mvalor(YPred_ant, img, red, signal_detected, detection_text, debounce)

% Dimensiones de la imagen
[x, y, z] = size(img);
% Se redimensiona la imagen si es necesario
if any([x, y, z] ~= [224, 224, 3])
    img = imresize(img, [224, 224]);
end

% Se analiza la imagen
[YPred, probs] = classify(red, Resnet50_mvalor, img);

cambio_signal = 0.7;
prob_max = max(probs);

% cambio deteccin y filtrado
if(prob_max >= cambio_signal)
    if (YPred_ant == YPred)
        if debounce < 12
            debounce = debounce + 1;
        end
    else
        %Cambio en la detección
        debounce = 0;
    end
else
    %Falta de certeza en la detección
    if debounce > 0
        debounce = debounce - 1;
    end
end

if (debounce >= 6 & YPred_ant == YPred)
    YPred_ant = YPred;
```

### Función detección\_signal\_mvalor

```
% La señal si ha pasado el control
detection_text = char(YPred) + ": " + num2str(max(probs)*100, '%.2f') + "%";
signal_detected = true;
else
  YPred_ant = YPred;
  % La señal no ha pasado el control
  YPred = "No se detectó ninguna señal con suficiente certeza.";
  detection_text = char(YPred);
  signal_detected = false;
end
end
```

## 1.8. Código para la actuación ante las señales

### Función actuador\_signal

```
function [detection_detected, speed, time, time_started] = actuador_signal(speed, YPred,
distanceToBar, img, detection_detected, client, time, time_started)
sim = client.getObject('sim');

switch YPred
  case 'Callejon'
    speed = frenado_to_0(distanceToBar);
    detection_detected = false;
  case 'Parking'
    detection_detected = false;
    speed = frenado(distanceToBar);
  case 'Paso de peatones'
    speed = 1.5;
  case 'Prohibido'
    detection_detected = false;
    speed = frenado_to_0(distanceToBar);
  case 'Rotonda'
    detection_detected = false;
    speed = frenado(distanceToBar);
  case 'Semaforo'
    detection_detected = false;
    type = detector_luz(img);
    if strcmp(type, 'rojo')
      speed = 0;
    elseif strcmp(type, 'ambar')
      speed = 0.6;
    elseif strcmp(type, 'verde')
      speed = 1.5;
    end
  case 'Stop'
    [detection_detected, speed, time, time_started] = detencion_stop (client,
detection_detected, distanceToBar, time, time_started);
  case 'Sentido único'
    detection_detected = false;
    speed = 1.5;
  case 'Limite velocidad 30'
    speed = 0.3;
    detection_detected = false;
  otherwise
    detection_detected = false;
    speed = 1.5;
end
end
```

## 1.9. Código de la función para la detección del carril

### Función detección\_carril\_mvalor

```
function [angleLaneLeft, angleLaneRight, centerLaneRight, bordes_carril, GPS_activated] =  
deteccion_carril_mvalor(IMG)  
% Aplicar las funciones de clasificación  
img_arcen = clasi_arcen(IMG);  
img_cielo = clasi_cielo(IMG);  
img_cesped = clasi_cesped(IMG);  
img_linea_divisoria = clasi_linea_divisoria(IMG);  
disk = strel("square",3);  
img_linea_divisoria = imdilate(img_linea_divisoria,disk);  
  
% Combinar las imágenes binarias usando la operación OR  
img_final = img_arcen | img_cielo | img_cesped | img_linea_divisoria;  
img_final = ~ img_final;  
% % Crear subplot de 2x3  
% figure;  
% subplot(2, 3, 1);  
% imshow(~img_final);  
% title('Imagen Final');  
%  
% subplot(2, 3, 2);  
% imshow(img_arcen);  
% title('Arcén Binario');  
%  
% subplot(2, 3, 3);  
% imshow(img_cielo);  
% title('Cielo Binario');  
%  
% subplot(2, 3, 4);  
% imshow(img_cesped);  
% title('Césped Binario');  
%  
% subplot(2, 3, 5);  
% imshow(img_linea_divisoria);  
% title('Linea divisoria binaria');  
  
% Eliminamos los bordes laterales para limar asperezas  
  
% Obtiene el tamaño de la imagen  
[h, w] = size(img_final);  
  
% Define el número de píxeles a eliminar de los bordes  
cut = 20;  
  
% Pone a 0 los píxeles de los lados excepto los 40 últimos de abajo que  
% forman parte de la carretera  
img_final(1:cut, :) = 0; % Arriba  
img_final(1:(h-110), 1:cut) = 0; % Izquierda  
img_final(1:(h-110), w-cut+1:w) = 0; % Derecha  
  
% Reducción a los dos objetos de mayor área  
img_final = Two_Biggest_Objects(img_final);  
  
% Se aplica imfill para rellenar huecos  
img_final = imfill(img_final,'holes');  
  
% Eliminar imperfecciones la linea divisoria  
stats_linea = regionprops(img_linea_divisoria, 'Area', 'PixelIdxList');  
  
% Obtener todas las áreas de los objetos  
areas_linea = [stats_linea.Area];  
if ~isempty(stats_linea)  
    % Obtener todas las áreas de los objetos  
    areas_linea = [stats_linea.Area];  
  
    % Encontrar el índice del objeto con el área más grande  
    [max_area_linea, max_idx_linea] = max(areas_linea);
```



## Función detección\_carril\_mvalor

```
% Crear una imagen vacía del mismo tamaño que img_linea_divisoria
img_linea_divisoria = zeros(size(img_linea_divisoria));

% Obtener la máscara del objeto con el área más grande
img_linea_divisoria(stats_linea(max_idx_linea).PixelIdxList) = 1;

% Cálculo del límite de la línea divisoria
h_lim = 0;
w_lim = 0;
count = 0;
[h, w] = size(img_linea_divisoria);
% Encontrar las coordenadas del píxel más alto con valor 1
for i=h:-1:1
    for j=w:-1:1
        % Si se detecta línea almacenar la altura de ese píxel
        if(1 == img_linea_divisoria(i,j))
            count = count + 1;
            if(count >= 5)
                count = 0;
                h_lim = i;
                w_lim = j;
            end
        end
    end
end
else
% En caso de no encontrar ninguna línea, asignar valores predeterminados
h_lim = 0;
w_lim = 0;
end

% Eliminar el carril de la izquierda
for i=1:h
    for j=w:-1:1
        % Si se detecta línea
        if(1 == img_linea_divisoria(i,j))
            while(j>1)
                img_final(i,j) = 0;
                j = j - 1;
            end
        end
    end
end

% Detección de los bordes del carril
img_bordes = bwareaopen(img_final,50);
bordes_carril = edge(img_bordes,'Canny');

% Recorte superior para adaptación a curvas
[h, w] = size(bordes_carril);

% Recortar de la parte superior de la imagen
bordes_carril(1:(h-15), :) = 0;

% Se obtienen los ángulos de los bordes del carril
[L, ~] = bwlabel(bordes_carril);
props = regionprops(L, 'Centroid', 'MajorAxisLength', 'Orientation');
[~, index] = sort([props.MajorAxisLength], 'descend');
filteredOrientations = [props(index).Orientation];
centerLaneRight = [0, 0];

dif_right = inf;
dif_left = inf;
angleLaneRight = 0;
angleLaneLeft = 0;

if numel(props) > 1
    % Robustez ante más de un ángulo por si hubiese errores y se detectase más
    % de un borde debido a un fallo en el edge
```

### Función detección\_carril\_mvalor

```
for i = 1:numel(filteredOrientations)

    % Filtrar y seleccionar el ángulo más cercano a -35 para el carril derecho
    if filteredOrientations(i) >= -85 && filteredOrientations(i) <= -5
        if abs(filteredOrientations(i) + 35) < abs(dif_right)
            angleLaneRight = filteredOrientations(i);
            dif_right = abs(filteredOrientations(i) + 35);

            % Calcular el centroide del carril derecho
            stats = regionprops(L, 'Centroid', 'Orientation');
            for j = 1:numel(stats)
                if stats(j).Orientation == angleLaneRight
                    centerLaneRight = stats(j).Centroid;
                    break; % Detener la iteración después de encontrar el centroide del
carril derecho
                end
            end
        end
    end

    % Filtrar y seleccionar el ángulo más cercano a 35 para el carril izquierdo
    if filteredOrientations(i) >= 5 && filteredOrientations(i) <= 85
        if abs(filteredOrientations(i) - 35) < abs(dif_left)
            angleLaneLeft = filteredOrientations(i);
            dif_left = abs(filteredOrientations(i) - 35);
        end
    end

    GPS_activated = false;
else

    % Se activa el GPS por haberse detectado un cruce o rotonda
    GPS_activated = true;
end

% imshow(IMG)
% hold on; % Mantener el gráfico actual
% plot(centerLaneRight(1), centerLaneRight(2), 'ro', 'MarkerSize', 5); % Plotear el centro del
carril derecho como un círculo rojo
% hold off; % Liberar el gráfico
end
```

## 1.10. Código completo para las pruebas de la detección del carril.

### Código para las pruebas de la detección del carril

```
clear all;
close all;

% addpath('Function/Carril/');
% addpath('Function/Señales/');
% addpath('Function/Path/');

%% ----- VARIABLES -----
% Nuevas
signal_detected = false;
GPS_Activated = false;
detection_text = "No se detectó ninguna señal.";
red = load('Deep Learning/Resnet50_mvalor.mat');
speed = 10;
YPred_ant = '';
```

## Código para las pruebas de la detección del carril

```
debounce = 0;
detection_detected = false;
time_started = 0;
GPS_activated = false;

% Antiguas
global LENGTH WIDTH IMG_DISPLAY;
LENGTH = 0.316;
WIDTH = 0.213;
areas = 0;
cteP = -0.01;
CteO = 100;

% -----
%% ----- INICIALIZACIÓN -----

client = RemoteAPIClient();
sim = client.getObject('sim');
fprintf('Program started\n');

visionSensorHandle = sim.getObject('/ViewCamera');
passiveVisionSensorHandle = sim.getObject('/ViewVisualization');

client.setStepping(true);
sim.startSimulation();

startTime = sim.getSimulationTime();
time = startTime;
client.step();

[gps, compass, steer, lights] =
sim.callScriptFunction('get_state@/Car',sim.scripttype_childscript);

% -----
%% ----- MAIN -----
while time - startTime < 100

    client.step();
    img = getImage(sim, visionSensorHandle);
    imshow(img)
    IMG_DISPLAY = img;
    [laneLeft, laneRight, centerLaneRight, bordes_carril, GPS_activated] =
deteccion_carril_mvaltor_test(img);

    [barDetected, distanceToBar] = detectBar(img);
    [debounce, detection_text, YPred, YPred_ant, signal_detected] =
deteccion_signal_mvaltor(YPred_ant, img, red, signal_detected, detection_text, debounce);

    IMG_DISPLAY = insertText(IMG_DISPLAY, [10, 10], string(detection_text), 'TextColor',
'black', 'FontSize', 14);
    IMG_DISPLAY = insertText(IMG_DISPLAY, [10, 40], ['Distance to bar: '
num2str(distanceToBar)], 'TextColor', 'black', 'FontSize', 14);
    IMG_DISPLAY = insertText(IMG_DISPLAY, [10, 70], ['Speed: ' num2str(speed)], 'TextColor',
'black', 'FontSize', 14);
    lightOption = 1;
    [leftAngle, rightAngle] = wheelControl(laneLeft, laneRight);
    setCarControl(sim, steer, lights, speed, leftAngle, rightAngle, lightOption);
    visualizaImagenCoppelia(IMG_DISPLAY, sim, passiveVisionSensorHandle);
    time = sim.getSimulationTime();
    close all
end

sim.stopSimulation();

fprintf('Program ended\n');
% -----
```

## Código para las pruebas de la detección del carril

```
%% ----- FUNCTIONS -----  
  
function setCarControl(sim, steer, lights, speed, leftAngle, rightAngle, lightOption)  
    steer{2}{1} = leftAngle;  
    steer{2}{2} = rightAngle;  
    steer{1} = speed;  
    lights{1} = lightOption;  
    data = {steer, lights};  
    sim.callScriptFunction('set_state@/Car',sim.scripttype_childscript, data);  
end  
  
function img = getImage(sim, handle)  
    [img, resX, resY] = sim.getVisionSensorCharImage(handle);  
    img = flip(permute(reshape(img, 3, resX, resY), [3 2 1]), 1);  
end  
  
function visualizaImagenCoppelia(img, sim, handle)  
    img = reshape(permute(flip(img, 1), [3 2 1]), [1 width(img) * height(img) * 3]);  
    sim.setVisionSensorCharImage(handle, img);  
end  
  
function [left_angle, right_angle] = wheelControl(laneLeft, laneRight)  
    global LENGTH WIDTH;  
  
    % Estandar (recta)  
    angle = -(laneLeft + laneRight);  
    if angle ~= 0  
        radius = LENGTH / tan(deg2rad(angle));  
    else  
        radius = Inf;  
    end  
    close_angle = atan(LENGTH / (abs(radius) - WIDTH/2));  
    far_angle = atan(LENGTH / (abs(radius) + WIDTH / 2));  
  
    if radius > 0  
        % Giro hacia la izquierda  
        left_angle = far_angle;  
        right_angle = close_angle;  
    else  
        % Giro hacia la derecha  
        left_angle = -close_angle;  
        right_angle = -far_angle;  
    end  
end  
% -----
```

## 1.11. Código de la detección de la barra

### Deteccción de la barra

```
function [barDetected, distanceToBar] = detectBar(img)  
    % Tamaño de la barra (metros) y parámetros de la cámara e imagen  
    barCurrentSize = 0.48978;  
    AFOV = 80;  
    numPixV = 256;  
    barDetected = false;  
  
    img_linea_divisoria = clasi_linea_divisoria(img);  
  
    % Eliminar imperfecciones en la línea divisoria  
    stats_barra = regionprops(img_linea_divisoria, 'Area', 'PixelIdxList');  
  
    % Obtener todas las áreas de los objetos
```

## Detección de la barra

```
areas_linea = [stats_barra.Area];
if ~isempty(stats_barra)
    % Encontrar el índice del objeto con el área más grande
    [~, max_idx_linea] = max(areas_linea);
    % Crear una imagen vacía del mismo tamaño que img_linea_divisoria
    img_linea_divisoria = zeros(size(img_linea_divisoria));
    % Obtener la máscara del objeto con el área más grande
    img_linea_divisoria(stats_barra(max_idx_linea).PixelIdxList) = 1;

    % Cálculo del límite de la línea divisoria
    h_lim = 0;
    w_lim = 0;
    count = 0;
    [h, w] = size(img_linea_divisoria);
    % Encontrar las coordenadas del píxel más alto con valor 1
    for i = h:-1:1
        for j = w:-1:1
            % Si se detecta línea, almacenar la altura de ese píxel
            if img_linea_divisoria(i, j) == 1
                count = count + 1;
                if count >= 5
                    count = 0;
                    h_lim = i;
                    w_lim = j;
                end
            end
        end
    end
    % Establecer a 0 los píxeles a la izquierda de w_lim
    for i = h:-1:1
        for j = 1:1:w_lim
            img(i, j, :) = 0;
        end
    end
end

% Segmentar la imagen para detectar las barras
bw = segmentacion_barra(img);

% Morfología matemática para eliminar imperfecciones y quedarnos con la
% línea vertical
se = strel("rectangle", [10, 1]);
se_dilate = strel("square", 3);
bw_opened = imopen(bw, se);
bw_opened = imdilate(bw_opened, se_dilate);

% Etiquetar las regiones conectadas en la imagen binaria
[L, numRegions] = bwlabel(bw_opened);

% Extraer propiedades de las regiones etiquetadas
props = regionprops(L, 'MajorAxisLength', 'Orientation');

% Ordenar las regiones por longitud del eje principal en orden descendente
[~, index] = sort([props.MajorAxisLength], 'descend');

% Iterar sobre las regiones etiquetadas
for i = 1:numel(index)

    % Longitud del eje principal de la región etiquetada
    barMajorAxisLength = props(index(i)).MajorAxisLength;

    % Ángulo de orientación de la barra
    angle = props(index(i)).Orientation;

    % Condiciones para detectar la barra
    if abs(angle) > 80 && abs(angle) < 100 && barMajorAxisLength > 25
        % Marcador de detección de la barra
        distanceToBar = (numPixV * barCurrentSize) / (2 * barMajorAxisLength * tand(AFOV /
2));
        barDetected = true;
    end
end
```

### Detección de la barra

```
% Finalizar la iteración
break;
end
end

% Si no se detecta la barra, establecer la distancia como infinito
if ~barDetected
    distanceToBar = inf;
end
end
```

## 1.12. Código completo para mostrar la ruta óptima

### Función ruta\_seleccionada

```
function ruta_seleccionada(ListaPaths, GPS)
% Crear un nuevo gráfico para las rutas seleccionadas
figure;
hold on;

% Inicializar variables para la leyenda
legend_entries = {};
legend_labels = {};

% Iterar sobre cada elemento de ListaPaths
for i = 1:length(ListaPaths)
    % Obtener información de la ruta actual
    ruta_actual = ListaPaths{i};

    % Extraer las coordenadas de los puntos de inicio y fin de la ruta
    start_point = ruta_actual.start_point;
    end_point = ruta_actual.end_point;

    % Dibujar los puntos de inicio y fin de la ruta
    plot(cell2mat(start_point(1)), cell2mat(start_point(2)), 'ro', 'MarkerSize', 3); % Punto de inicio
    plot(cell2mat(end_point(1)), cell2mat(end_point(2)), 'go', 'MarkerSize', 3); % Punto de fin

    % Añadir los puntos de inicio y fin a la leyenda si es la primera ruta
    if i == 1
        legend_entries{end+1} = plot(NaN, NaN, 'ro', 'MarkerSize', 3);
        legend_labels{end+1} = 'Puntos de Inicio';
        legend_entries{end+1} = plot(NaN, NaN, 'go', 'MarkerSize', 3);
        legend_labels{end+1} = 'Punto de Fin';
    end

    % Dibujar los segmentos de la ruta
    puntos = ruta_actual.puntos;
    for j = 1:length(puntos)-1
        point1 = puntos{j};
        point2 = puntos{j+1};
        plot([cell2mat(point1(1)) cell2mat(point2(1))], [cell2mat(point1(2))
cell2mat(point2(2))], 'c-', 'LineWidth', 1.2);
    end

    % Plotear la posición GPS como 'x'
    plot(cell2mat(GPS(1)), cell2mat(GPS(2)), 'bx', 'MarkerSize', 5);

    % Añadir la posición GPS a la leyenda
    legend_entries{end+1} = plot(NaN, NaN, 'bx', 'MarkerSize', 5);
    legend_labels{end+1} = 'Posición GPS';

    % Añadir segmentos de la ruta a la leyenda
    legend_entries{end+1} = plot(NaN, NaN, 'c-', 'LineWidth', 1.2);
```

### Función ruta\_seleccionada

```
legend_labels{end+1} = 'Segmentos de la Trayectoria';  
  
% Configurar la leyenda  
legend([legend_entries{:}], legend_labels, 'Location', 'best');  
  
hold off;  
title('Trayectoria más corta');  
xlabel('Coordenada X');  
ylabel('Coordenada Y');  
end
```

## 1.13. Código completo para el seguimiento del GPS

### Función wheelControlFromRoute

```
function [left_angle, right_angle] = wheelControlFromRoute(ListaPaths, GPS, ruta_actual,  
compass)  
    LENGTH = 0.316;  
    WIDTH = 0.213;  
  
    % Obtener los puntos de la ruta más cercana  
    puntos = ruta_actual.puntos;  
  
    % Inicializar la distancia mínima y el índice del punto más cercano  
    distancia_minima = Inf;  
    indice_punto_cercano = 0;  
  
    % Calcular la distancia entre el GPS y cada punto en la ruta  
    for j = 1:length(puntos)  
        punto = puntos{j};  
        distancia = sqrt((punto{1} - GPS{1})^2 + (punto{2} - GPS{2})^2);  
        if distancia < distancia_minima  
            distancia_minima = distancia;  
            indice_punto_cercano = j;  
        end  
    end  
  
    % Asegurarse de que el índice objetivo no exceda el número de puntos  
    indice_punto_objetivo = min(indice_punto_cercano + 4, length(puntos));  
    punto_objetivo = puntos{indice_punto_objetivo};  
  
    % Obtener la diferencia en x y y entre el GPS y el punto objetivo  
    x_diff = punto_objetivo{1} - GPS{1};  
    y_diff = punto_objetivo{2} - GPS{2};  
  
    % Calcular el ángulo entre el GPS y el punto más cercano  
    angle = atan2(y_diff, x_diff) - compass;  
    angle = rad2deg(angle);  
  
    if angle ~= 0  
        radius = LENGTH / tan(deg2rad(angle));  
    else  
        radius = Inf;  
    end  
  
    close_angle = atan(LENGTH / (abs(radius) - WIDTH / 2));  
    far_angle = atan(LENGTH / (abs(radius) + WIDTH / 2));  
  
    if radius > 0  
        % Giro hacia la izquierda  
        left_angle = far_angle;  
        right_angle = close_angle;  
    else  
        % Giro hacia la derecha
```

### Función wheelControlFromRoute

```
left_angle = -close_angle;  
right_angle = -far_angle;  
end  
  
end
```

## 1.14. Código completo del script para las pruebas de GPS

### Código completo para las pruebas del GPS

```
clear all;  
close all;  
  
addpath('Function/Carril/');  
addpath('Function/Señales/');  
addpath('Function/Path/');  
  
%% ----- VARIABLES -----  
% Nuevas  
signal_detected = false;  
doble_filtro = 0;  
status = 'recta';  
detection_text = "No se detectó ninguna señal.";  
red = load('Deep Learning/Resnet50_mvaltor.mat');  
giro = [0 45];  
bool_curva = false;  
speed = 0.2;  
lightOption = 1;  
  
% Antiguas  
global TURNING_ANGLE LENGTH WIDTH right_angle left_angle Compass GPS Velocidad;  
TURNING_ANGLE = pi/3;  
LENGTH = 0.316;  
WIDTH = 0.213;  
Velocidad = speed;  
ACCEPTABLE_POINTS_DISTANCE = 0.01;  
  
% ----- INICIALIZACIÓN -----  
  
client = RemoteAPIClient();  
sim = client.getObject('sim');  
fprintf('Program started\n');  
  
visionSensorHandle = sim.getObject('/ViewCamera');  
passiveVisionSensorHandle = sim.getObject('/ViewVisualization');  
  
client.setStepping(true);  
sim.startSimulation();  
  
startTime = sim.getSimulationTime();  
t = startTime;  
client.step();  
  
[gps, compass, steer, lights] =  
sim.callScriptFunction('get_state@/Car',sim.scripttype_childscript);  
  
meta = sim.callScriptFunction('get_meta@/Meta',sim.scripttype_childscript);  
[roundabouts_meta, streets_meta, crossings_meta, parkings_meta] = meta{:};
```



## Código completo para las pruebas del GPS

```
% -----  
%% -----Antiguo-----  
  
paths = path(streets_meta, roundabouts_meta, crossings_meta, parkings_meta);  
[Origen, Destino, weights] = clasi_paths(paths,ACCEPTABLE_POINTS_DISTANCE);  
  
% Crear el gráfico con las combinaciones y los pesos  
G = digraph(Origen, Destino, weights);  
%  
% % Primer subplot: gráfico con pesos en las aristas  
% subplot(1, 2, 1);  
% plot(G, 'EdgeLabel', G.Edges.Weight);  
% title('Combinaciones posibles con pesos');  
%  
% % Segundo subplot: gráfico sin pesos en las aristas  
% subplot(1, 2, 2);  
% plot(G);  
% title('Combinaciones posibles sin pesos');  
  
% RutaOrigen = obtenerOrigen(paths, gps);  
% RutaDestino = obtenerDestino(paths);  
  
PathOrigen = getInfo(RutaOrigen, paths);  
PathDestino = getInfo(RutaDestino, paths);  
  
[P,d] = shortestpath(G, PathOrigen, PathDestino);  
ListaPaths = getPaths(P, paths);  
  
ruta_seleccionada(ListaPaths, gps);  
%% -----  
while t - startTime < 100  
    client.step();  
    [img, resX, resY] = sim.getVisionSensorCharImage(visionSensorHandle);  
    img_Buena = flip(permute(reshape(img, 3, resX, resY), [3 2 1]), 1);  
  
    left_angle = 0;  
    right_angle = 0;  
  
    [gps, compass, steer, lights] =  
    sim.callScriptFunction('get_state@/Car',sim.scripptype_childscript);  
  
    Compass = compass;  
    GPS = gps;  
  
    sim.setVisionSensorCharImage(passiveVisionSensorHandle, img);  
  
    % Obtener la ruta más cercana  
    ruta_actual = ruta_cercana(ListaPaths, GPS);  
  
    [left_angle_gps, right_angle_gps] = wheelControlFromRoute(ListaPaths, GPS, ruta_actual,  
    compass);  
    setCarControl(sim, steer, lights, speed, left_angle_gps, right_angle_gps, lightOption);  
  
    t = sim.getSimulationTime();  
end  
  
sim.stopSimulation();  
  
fprintf('Program ended\n');  
  
function setCarControl(sim, steer, lights, speed, leftAngle, rightAngle, lightOption)  
    steer{2}{1} = leftAngle;  
    steer{2}{2} = rightAngle;
```

### Código completo para las pruebas del GPS

```
steer{1} = speed;
lights{1} = lightOption;
data = {steer, lights};
sim.callScriptFunction('set_state@/Car',sim.scripttype_childscript, data);
end
```

## 1.15. Código completo del script principal de integración

### Código de la implementación final

```
% -----IMPLEMENTACIÓN CONJUNTA-----
clear all;
close all;

addpath('Function/Carril/');
addpath('Function/Señales/');
addpath('Function/Path/');

% ----- VARIABLES -----
signal_detected = false;

detection_text = "No se detectó ninguna señal.";
red = load('Deep Learning/Resnet50_mvaltor.mat');
speed = 1.5;
lightOption = 1;
time_started = 0;
YPred_ant = '';
debounce = 0;
GPS_activated = false;
detection_detected = false;
GPS_trayectoria_X = [];
GPS_trayectoria_Y = [];
ACCEPTABLE_POINTS_DISTANCE = 0.01;

% ----- INICIALIZACIÓN SIMULACIÓN -----
client = RemoteAPIClient();
sim = client.getObject('sim');
fprintf('Program started\n');

visionSensorHandle = sim.getObject('/ViewCamera');

passiveVisionSensorHandle = sim.getObject('/ViewVisualization');
passiveVisionSensorHandle_aux = sim.getObject('/ViewVisualization_aux');

client.setStepping(true);
sim.startSimulation();

time_started = sim.getSimulationTime();
t = time_started;
client.step();

[gps, compass, steer, lights] =
sim.callScriptFunction('get_state@/Car',sim.scripttype_childscript);
meta = sim.callScriptFunction('get_meta@/Meta',sim.scripttype_childscript);

% ----- INICIALIZACIÓN PATH -----
```

## Código de la implementación final

```
[roundabouts_meta, streets_meta, crossings_meta, parkings_meta] = meta{:};

paths = path(streets_meta, roundabouts_meta, crossings_meta, parkings_meta);

[Origen, Destino, weights] = classi_paths(paths,ACCEPTABLE_POINTS_DISTANCE);

% Crear el gráfico con las combinaciones y los pesos
G = digraph(Origen, Destino, weights);

% Primer subplot: gráfico con pesos en las aristas
subplot(1, 2, 1);
plot(G, 'EdgeLabel', G.Edges.Weight);
title('Combinaciones posibles con pesos');

% Segundo subplot: gráfico sin pesos en las aristas
subplot(1, 2, 2);
plot(G);
title('Combinaciones posibles sin pesos');

RutaOrigen = obtenerOrigen(paths, gps);
RutaDestino = obtenerDestino(paths);

PathOrigen = getInfo(RutaOrigen, paths);
PathDestino = getInfo(RutaDestino, paths);

[P,d] = shortestpath(G, PathOrigen, PathDestino);
ListaPaths = getPaths(P, paths);

ruta_seleccionada(ListaPaths, gps);

% -----
%% ----- MAIN -----
while true

    client.step();
    [gps, compass, steer, lights] =
sim.callScriptFunction('get_state@/Car',sim.scripptype_childscript);
    img = getImage(sim, visionSensorHandle);
    IMG_DISPLAY = img;

    % Actualizar las variables de trayectoria con las coordenadas actuales
    GPS_trayectoria_X = [GPS_trayectoria_X, gps{1}];
    GPS_trayectoria_Y = [GPS_trayectoria_Y, gps{2}];

    if GPS_activated
        % Obtener la ruta más cercana
        ruta_actual = ruta_cercana(ListaPaths, gps);

        % Se reduce la velocidad por ser un tramo de curvas
        speed = 0.2;
        [leftAngle, rightAngle] = wheelControlFromRoute(ListaPaths, gps, ruta_actual,
compass);
        [~, ~, ~, bordes_carril, GPS_activated] = deteccion_carril_mvalor(img);
        modo = 'GPS';
    else
        [laneLeft, laneRight, centerLaneRight, bordes_carril, GPS_activated] =
deteccion_carril_mvalor(img);
        [barDetected, distanceToBar] = detectBar(img);
        [leftAngle, rightAngle] = wheelControl(laneLeft, laneRight);
        speed = aceleracion(laneLeft, laneRight);
        modo = 'Siguelineas';
    end

    [debounce, detection_text, YPred, YPred_ant, signal_detected] =
deteccion_signal_mvalor(YPred_ant, img, red, signal_detected, detection_text, debounce);
    if (signal_detected & ~GPS_activated)
        [detection_detected, speed, time, time_started] = actuador_signal(speed, YPred,
distanceToBar, img, detection_detected, client, time, time_started);
```

## Código de la implementación final

```

end

    IMG_DISPLAY = insertText(IMG_DISPLAY, [10, 10], string(detection_text), 'TextColor',
'black', 'FontSize', 14);
    IMG_DISPLAY = insertText(IMG_DISPLAY, [10, 40], ['Distance to bar: '
num2str(distanceToBar)], 'TextColor', 'black', 'FontSize', 14);
    IMG_DISPLAY = insertText(IMG_DISPLAY, [10, 70], ['Speed: ' num2str(speed)], 'TextColor',
'black', 'FontSize', 14);

    bordes_carril_rgb = uint8(repmat(bordes_carril, [1, 1, 3]) * 255);

    bordes_carril_rgb = insertText(bordes_carril_rgb, [10, 10], ['Modo actual: ' modo],
'TextColor', 'black', 'FontSize', 14);
    bordes_carril_rgb = insertText(bordes_carril_rgb, [10, 40], ['Ángulo izquierdo: '
num2str(leftAngle)], 'TextColor', 'black', 'FontSize', 14);
    bordes_carril_rgb = insertText(bordes_carril_rgb, [10, 70], ['Ángulo derecho: '
num2str(rightAngle)], 'TextColor', 'black', 'FontSize', 14);

    setCarControl(sim, steer, lights, speed, leftAngle, rightAngle, lightOption);

    visualizaImagenCoppelia(IMG_DISPLAY, sim, passiveVisionSensorHandle);
    visualizaImagenCoppelia(bordes_carril_rgb, sim, passiveVisionSensorHandle_aux);

    time = sim.getSimulationTime();

    % Revisa si el vehículo ha completado el trayecto
    x_diff = ListaPaths{length(ListaPaths)}.end_point{1} - gps{1};
    y_diff = ListaPaths{length(ListaPaths)}.end_point{2} - gps{2};
    distance = sqrt(x_diff^2 + y_diff^2);

    % Se ha llegado al destino
    if distance <= ACCEPTABLE_POINTS_DISTANCE
        disp('Se ha completado el trayecto')
        break;
    end
end

end

sim.stopSimulation();

% Graficar la trayectoria realizada
figure;
ruta_seleccionada(ListaPaths, gps);
hold on
plot(GPS_trayectoria_X, GPS_trayectoria_Y, 'b--', 'LineWidth', 1.5, 'DisplayName', 'Trayectoria
real');
title('Trayecto completado');
xlabel('Posición X');
ylabel('Posición Y');
hold off

% -----
%% ----- FUNCIONES -----
function setCarControl(sim, steer, lights, speed, leftAngle, rightAngle, lightOption)
    steer{2}{1} = leftAngle;
    steer{2}{2} = rightAngle;
    steer{1} = speed;
    lights{1} = lightOption;
    data = {steer, lights};
    sim.callScriptFunction('set_state@/Car', sim.scripttype_childscript, data);
end

function img = getImage(sim, handle)
    [img, resX, resY] = sim.getVisionSensorCharImage(handle);
    img = flip(permute(reshape(img, 3, resX, resY), [3 2 1]), 1);
end

function visualizaImagenCoppelia(img, sim, handle)

```

### Código de la implementación final

```
img = reshape(permute(flip(img, 1), [3 2 1]), [1 width(img) * height(img) * 3]);
sim.setVisionSensorCharImage(handle, img);
end
% -----
```

## 2. Agenda 2030, ODS

En la siguiente tabla se muestra la relación de este proyecto con los objetivos de desarrollo sostenible de la agenda 2030.

| Objetivos de desarrollo sostenible         | Alto | Medio | Bajo | No procede |
|--|------|-------|------|------------|
| 1. Fin de la pobreza                       |      |       |      | X          |
| 2. Hambre cero                             |      |       |      | X          |
| 3. Salud y bienestar                       |      |       |      | X          |
| 4. Educación de calidad                    | X    |       |      |            |
| 5. Igualdad de género                      |      |       |      | X          |
| 6. Agua limpia y saneamiento               |      |       |      | X          |
| 7. Energía asequible y limpia              |      |       |      | X          |
| 8. Trabajo decente y crecimiento económico |      |       |      | X          |
| 9. Industria, innovación e infraestructura | X    |       |      |            |
| 10. Reducción de las desigualdades         |      |       |      | X          |
| 11. Ciudades y comunidades sostenibles     | X    |       |      |            |
| 12. Producción y consumo responsables      |      |       |      | X          |
| 13. Acción por el clima                    |      | X     |      |            |
| 14. Vida submarina                         |      |       |      | X          |
| 15. Vida de ecosistemas terrestres         |      |       |      | X          |
| 16. Paz, justicia e instituciones sólidas  |      |       |      | X          |
| 17. Alianzas para lograr los objetivos     | X    |       |      |            |

Tabla I. Tabla de los objetivos de desarrollo sostenible

- **ODS 4. Educación de calidad**

El proyecto contribuye a la formación en el campo de la conducción autónoma así como en técnicas de visión por computación o en la planificación y seguimiento de trayectorias.

- **ODS 9. Industria, innovación e infraestructura**

El proyecto se centra en el desarrollo de un sistema innovador en uno de los campos más novedosos como es el de la conducción autónoma o las redes neuronales convolucionales.

- **ODS 11. Ciudades y comunidades sostenibles**

Además de mejorar la movilidad urbana y la eficiencia del transporte dentro de las ciudades la conducción autónoma reduce el consumo de los vehículo al anticiparse y reaccionar antes ante obstáculos.

- **ODS 13. Acción por el clima**

Al optimizar las rutas tomadas y mejorar la movilidad urbana se reducirán los gases contaminantes emitidos por los vehículos.

- **ODS 17. Alianzas para lograr los objetivos**

Se combinan distintas técnicas y métodos como las redes neuronales, la visión por computación, la planificación de trayectorias o el control de un vehículo Ackerman para el cumplimiento de objetivos complejos e innovadores.



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



# UNIVERSITAT POLITÈCNICA DE VALÈNCIA

**Escuela Técnica Superior de Ingeniería del Diseño**

---

## **SISTEMA DE VISIÓN INTELIGENTE PARA EL GUIADO DE UN VEHÍCULO AUTÓNOMO EN UN ENTORNO URBANO**

**TRABAJO FINAL DE GRADO**

**GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL  
Y AUTOMÁTICA**

**Documento**

**Pliego de condiciones**

# Pliego de condiciones

---

|  |     |
|--|-----|
| 1. Objeto.....                                       | 167 |
| 2. Condiciones generales.....                        | 167 |
| 2.1. Pliego de condiciones facultativas .....        | 167 |
| 2.1.1. Obligaciones y derechos del proyectista ..... | 167 |
| 2.1.2. Planificación y tiempo de ejecución.....      | 167 |
| 2.1.3. Aceptación del proyecto .....                 | 168 |
| 2.2. Pliego de condiciones económicas.....           | 168 |
| 2.2.1. Fianzas y anticipos .....                     | 168 |
| 2.2.2. Composición de precios .....                  | 168 |
| 2.2.3. Mejoras del proyecto.....                     | 168 |
| 2.3. Pliego de condiciones legales.....              | 168 |



## 1. Objeto

El objeto de este documento es la fijación y aclaración de las condiciones técnicas mínimas sobre las cuales se podrá llevar a cabo este proyecto a través de los softwares Matlab y Coppelia sim para el control autónomo de un vehículo en un entorno urbano. Para su cumplimiento, se especifican las características necesarias para la realización de esta actividad:

- Garantizar la seguridad de todas aquellas personas involucradas en el proyecto de manera directa o indirecta
- Mejorar la calidad y la ejecución del proyecto
- Descripción precisa de los materiales empleados para la realización de este.

El uso adecuado de la simulación y el cumplimiento de este pliego de condiciones aseguran que se cumplan las especificaciones establecidas por el promotor. Es responsabilidad del ejecutor final aplicar correctamente la información proporcionada y hacerlo siempre de acuerdo con las condiciones estipuladas en el pliego, eximiendo al proyectista de cualquier responsabilidad por el incumplimiento de lo establecido en este documento.

## 2. Condiciones generales

En el presente documento se recopila las exigencias técnicas y legales para la correcta implementación de este proyecto.

### 2.1. Pliego de condiciones facultativas

En esta sección se detallan las condiciones específicas que deben cumplirse en la ejecución del proyecto de simulación. Estas condiciones están diseñadas para garantizar que tanto el proyectista como el promotor entiendan sus obligaciones y derechos, así como los tiempos de ejecución y el proceso de aceptación del proyecto.

A continuación, se describen las responsabilidades del proyectista, la planificación y los tiempos de ejecución, y las condiciones para la aceptación del proyecto por parte del cliente.

#### 2.1.1. Obligaciones y derechos del proyectista

El proyectista no se hace responsable de cualquier error causado por la modificación de una o varias partes del código por parte del consumidor, así mismo, el promotor será el encargado de facilitar todo el material necesario para la realización de este proyecto incluyendo tanto los materiales físicos como los softwares necesarios.

#### 2.1.2. Planificación y tiempo de ejecución

Se estipulará como fecha de inicio del proyecto aquella en la cual todos los documentos necesarios para justificar la rentabilidad de este estén firmados por el proyectista, el cual será el encargado de definir los ritmos de programación. Tras la firma de los documentos mencionados

con anterioridad, el proyectista contará con 5 días hábiles para la presentación de un ciclograma donde queden definidos los tiempos de ejecución del proyecto.

### 2.1.3. Aceptación del proyecto

La parte contratante contará con un plazo de 15 días hábiles durante la cual podrá solicitar cualquier tipo de modificación, pasado ese mes la solicitud de cualquier mejora o corrección no se considerará como parte del proyecto y por lo tanto su realización supondrá un coste extra.

## 2.2. Pliego de condiciones económicas

En el precio del proyecto se incluye el diseño y el coste de todos aquellos materiales utilizados para la realización de este, tanto la parte contratante como contratista se comprometen a cumplir con las cláusulas financieras de cada uno de ellos.

### 2.2.1. Fianzas y anticipos

El cliente deberá hacerse cargo de los siguientes pagos:

- **En la aceptación del proyecto:** Antes de iniciar con la ejecución del proyecto el cliente deberá abonar un 25% del coste total del proyecto.
- **Durante el proyecto:** En caso de que el proyectista haya cumplido con los plazos estipulados en el ciclograma se irá abonando un 50% del coste total dividido en cuotas mensuales.
- **Una vez se ha finalizado el proyecto:** Tras la entrega del proyecto completamente terminado, se abona el 25% restante.

En caso de demora el proyectista tendrá derecho al cobro extra de un 5% de la deuda por cada semana de retraso, del modo contrario, en caso de que el retraso se produzca por la parte contratista el cliente tendrá derecho a una reducción del 5% por semana de retraso.

### 2.2.2. Composición de precios

Los precios aplicados al proyecto están sujetos a las condiciones de los acuerdos aprobados entre el cliente y el proyectista.

### 2.2.3. Mejoras del proyecto

Cualquier modificación o ampliación del proyecto implicará un coste extra que deberá ser asumido por la parte contratante.

## 2.3. Pliego de condiciones legales

Las condiciones legales de este proyecto son las siguientes:



- **Marcas Registradas:** El proyectista y el promotor reconocen públicamente las marcas registradas que sean utilizadas durante el desarrollo y ejecución del proyecto.
- **Licencias:** El proyectista y el promotor reconocen tener al día las licencias del software utilizado.
- **Derechos de Autor:** Los derechos de autor de este proyecto se regirán por la legislación y reglamentación vigente al inicio del desarrollo del proyecto.



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



# UNIVERSITAT POLITÈCNICA DE VALÈNCIA

**Escuela Técnica Superior de Ingeniería del Diseño**

---

## **SISTEMA DE VISIÓN INTELIGENTE PARA EL GUIADO DE UN VEHÍCULO AUTÓNOMO EN UN ENTORNO URBANO**

**TRABAJO FINAL DE GRADO**

**GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL  
Y AUTOMÁTICA**

**Documento**

**Presupuesto**

# Presupuesto

---

|   |     |
|---|-----|
| 1. Cuadro de precios elementales .....  | 172 |
| 2. Cuadro de precios descompuestos..... | 172 |
| 3. Resumen.....                         | 172 |

## 1. Cuadro de precios elementales

Descomposición del presupuesto según los costes unitarios de los componentes del proyecto incluido materiales, licencias y mano de obra.

| Referencia          | Unidades | Denominación                  | Precio (€) |
|---------------------|----------|-------------------------------|------------|
| <b>Licencias</b>    |          |                               |            |
| l1                  | ud.      | CoppeliaSim Pro               | 2.145,00 € |
| l2                  | ud.      | Matlab                        | 900,00 €   |
| <b>Mano de obra</b> |          |                               |            |
| h1                  | h.       | Ingeniero de software         | 15,00 €    |
| h2                  | h.       | Equipo redactor de la memoria | 12,00 €    |
| <b>Materiales</b>   |          |                               |            |
| m1                  | ud.      | Portátil HP Intel core i5     | 750,00 €   |

*Tabla 1. Tabla de los cuadros de precios elementales*

## 2. Cuadro de precios descompuestos

Cuadro con los precios descompuestos en cada parte del proyecto incluyendo las cantidades para la realización de este en sus respectivas unidades y la amortización del material.

| Referencia   | Unidades | Denominación                  | Precio (€) | Cantidad | Total (€)         |
|--|----------|-------------------------------|------------|----------|-------------------|
| <b>Licencias</b>                                   |          |                               |            |          |                   |
| l1   | ud.      | CoppeliaSim Pro               | 2.145,00 € | 1        | 2.145,00 €        |
| l2   | ud.      | Matlab                        | 900,00 €   | 1        | 900,00 €          |
| <b>Total de las licencias</b>                      |          |                               |            |          | 3.045,00 €        |
| <b>Mano de obra</b>                                |          |                               |            |          |                   |
| h1   | h.       | Ingeniero de software         | 15,00 €    | 300      | 4.500,00 €        |
| h2   | h.       | Equipo redactor de la memoria | 12,00 €    | 75       | 900,00 €          |
| <b>Total de la mano de obra</b>                    |          |                               |            |          | 5.400,00 €        |
| <b>Materiales</b>                                  |          |                               |            |          |                   |
| m1   | ud.      | Portátil HP Intel core i5     | 750,00 €   | 0,25     | 187,50 €          |
| <b>Total de los materiales</b>                     |          |                               |            |          | 187,50 €          |
| <b>Presupuesto total del diseño y programación</b> |          |                               |            |          | <b>8.632,50 €</b> |

*Tabla 2. Tabla de los cuadros de precios descompuestos*

## 3. Resumen

Tabla final con el coste total del proyecto una vez aplicados los costes indirectos de este junto con el IVA aplicable.

| Denominación  | Total (€)          |
|---|--------------------|
| Diseño y programación                                   | 8.632,50 €         |
| Costes indirectos (10%)                                 | 863,25 €           |
| IVA (21%)   | 1.812,83 €         |
| <b>Presupuesto total para la ejecución del proyecto</b> | <b>11.308,58 €</b> |

*Tabla 3. Tabla resumen con el coste total del proyecto*