Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

**MÁSTER UNIVERSITARIO EN INGENIERÍA DE CONTROL AUTOMATIZACIÓN Y ROBÓTICA**

# TRABAJO FIN DE MÁSTER

## *DESARROLLO DE REDES NEURONALES CONVOLUCIONALES PARA ALGORITMOS DE NAVEGACIÓN*

**Estudiante** *Sánchez Chica, Ander*
**Director/Directora** *Zulueta Guerrero, Ekaitz*
**Departamento** Ing. De Sistemas y Automática
**Curso académico** *2020-2021*

*Bilbao, 24 de Junio de 2021*

# Contenido

## Resumen

Un AGV *(Automated Guided Vehicle)* es un robot móvil inteligente ampliamente utilizado para mover objetos o realizar tareas en diversos ámbitos como el industrial, los puertos, los almacenes o las zonas de trabajo peligrosas en las que el ser humano tendría importantes dificultades para trabajar. Dentro de las tecnologías que envuelve un AGV la navegación es clave.

Entre las diferentes alternativas existentes, para solucionar el problema de la navegación, los sistemas basados en visión están tomando especial relevancia en los últimos años, gracias en gran medida al aumento de la capacidad de computación que proporciona el uso de las GPUs (Graphics processing unit).

Los sistemas de navegación basados en visión emplean cámaras como sensor de entrada. Las cámaras son más fiables, más baratas y capaces de proporcionar una gran cantidad de información espacial. Además, la información extraída de la cámara puede utilizarse para la servo-orientación visual, la estimación del estado, la evitación de obstáculos y la planificación de la trayectoria.

Las Redes Neuronales Convolucionales, (CNN) por sus siglas en inglés, se han empleado ampliamente en el dominio de la imagen, mejorando significativamente el rendimiento de la clasificación de imágenes, la detección de objetos, la clasificación de escenas, etc. Por esto representan una herramienta de gran capacidad para tratar las imágenes y dar solución al problema de navegación.

En este trabajo se ha propuesto un modelo basado en redes neuronales convolucionales para solucionar el problema de navegación de un prototipo de AGV. Aplicando una clasificación basada en la segmentación semántica, se detecta una línea marcada en el suelo que representa la trayectoria a seguir por el robot. Emulando los sistemas filoguiados magnéticos existentes en la actualidad. La respuesta obtenida tras el procesamiento de la imagen se traduce en consignas al robot.

Adicionalmente se ha desarrollado un sistema de comunicación entre el PC que realiza el procesado de la imagen y el microcontrolador que maneja los motores del robot. También se ha realizado el cableado y programación del microcontrolador.



## Palabras Clave

AGV, Industria 4.0, Visión Artificial, Redes Neuronales Convolucionales (CNN), Deep Learning, Machine Learning, Filoguiado, GPUs.

# Índice de Figuras

# Índice de Tablas



# Acrónimos

Automated Guided Vehicle (AGVs)

Redes Neuronales Convolucionales (CNN)

Sistemas ciberfísicos (CPS)

Internet de las cosas (IOT)

Sensor de detección y alcance de luz (LiDAR)

Localización y el mapeo simultaneo (SLAM)

Inteligencia Artificial (IA)

Machine Learning (ML)

Redes neuronales recurrentes (RNN)

Deep Belief Neural Network (DBN)

Autoencoders apilados (SAE)

Región de interés (ROI)

Máquinas de Boltzmann restringidas (RBM)

Multi Layer Perceptron (MLP)

Fully convolutional Network (FCN)

Campos aleatorios condicionales (CRF)

Spatial Temporal Lane detection Network (STLNet)

Imágenes por segundo (fps)

Corriente continua (DC)

Rutina de servicio de interrupción (RSI)

# 1 Introducción y Contexto

## 1.1 Contexto Actual

El presente proyecto se desarrolla en el laboratorio de proyectos del departamento de Ingeniería de sistemas y automática, en Vitoria-Gasteiz. Se realiza como trabajo final de master para la titulación de Ingeniería de Control Robótica y automatización impartida en la escuela de ingeniería de Bilbao.

## 1.2 Introducción

La realización del proyecto viene motivada por la creciente evolución de los sistemas inteligentes en todas las áreas de la Industria. Debido al nuevo paradigma introducido por estándares como la Industria 4.0 o el Internet de las cosas, la presencia de sistemas inteligentes basados en tecnologías como la visión artificial están cobrando mayor peso.

El desarrollo de sistemas de navegación autónoma, que no dependan de una unidad de control central y permitan un manejo descentralizado de los AGVs (Automated Guided Vehicle) es uno de los principales retos ante los que se encuentra la industria.

Este proyecto busca resolver el problema de navegación para un prototipo de AGV. Para ello se va a desarrollar un sistema basado en visión artificial, que emplee CNN (Redes Neuronales Convolucionales) y permita al Robot navegar de forma autónoma.

El sistema de navegación desarrollado permite el movimiento de un prototipo de AGV a lo largo de una trayectoria delimitada por líneas marcadas en el suelo. Este sistema emula a los sistemas filoguiados basados en magnetismo presentes en la actualidad en diversas industrias. Sin embargo, el modelo propuesto permite mayor flexibilidad a la hora del rediseño de las trayectorias.

Para el desarrollo del sistema de navegación se ha acoplado una cámara al prototipo de AGV. En paralelo se ha acondicionado el hardware del prototipo para que sea capaz de manejar los motores ante las consignas comunicadas vía puerto serie al microcontrolador.

Los modelos de visión artificial han sido desarrollados empleando Matlab como plataforma de desarrollo. A sí mismo, el procesamiento de las imágenes capturadas por la cámara se realiza desde un PC colocado sobre el AGV. Es este mismo PC es el encargado de enviar las consignas al microcontrolador.

Se han empleado microcontroladores de Arduino para el control de los motores del robot y la comunicación con el PC de procesamiento. Se han utilizado los modelos UNO y Mega.

Finalmente, las pruebas de funcionamiento del prototipo se han realizado en el propio laboratorio de proyectos y en los pasillos adyacentes. Para ello se ha trazado una serie de líneas en el suelo que representen la trayectoria a seguir por el robot.

## 1.3 Estructura de la memoria

El contexto y la introducción mencionados previamente forman el primer capítulo de la memoria del proyecto. Seguidamente, se presenta el segundo capítulo *Alcance y objetivos* donde se

definen el ámbito y los hitos superados para el desarrollo del proyecto. A continuación, se presenta el tercer capítulo, *Estado del Arte*, donde se realiza una revisión bibliográfica de los conceptos y tecnologías principales del proyecto, como las CNN. El cuarto capítulo, *Desarrollo de la solución*, muestra los procedimientos seguidos para el desarrollo y entrenamiento de los modelos, así como en el desarrollo del hardware del AGV. En *Pruebas y Resultados*, quinto capítulo, se discuten los resultados obtenidos en las pruebas propuestas y se validan los modelos desarrollados. En el sexto capítulo *Conclusiones y Trabajos futuros*, se recapitulan los resultados obtenidos y se exploran alternativas o nuevas vías que tienen el potencial de mejorar y continuar el trabajo propuesto. Finalmente, en el último capítulo, se recogen las referencias bibliográficas citadas a lo largo del documento.

# 2 Alcance y Objetivos

## 2.1 Alcance

El proyecto busca resolver el problema de navegación para un vehículo de interiores. La solución desarrollada sirve como ejemplo de una navegación filoguiada basada en visión que permite al AGV moverse por una trayectoria predefinida y marcada en el suelo.

Las tecnologías desarrolladas para dar solución a este problema en particular son extrapolables a otros del mismo ámbito. De la misma forma que se desarrollan las redes convolucionales para la detección de una línea y utilizarla como trayectoria, se pueden emplear para destacar la superficie navegable por el robot y posteriormente aplicar un algoritmo que determine la ruta óptima.

El sistema de comunicación entre el PC y el microcontrolador permite que el AGV sea controlado por otras técnicas de navegación o equipos siempre que se mantenga la estructura de envío de consignas.

En conclusión, en la solución presentada en el proyecto se soluciona la navegación de un prototipo de AGV concreto empleando una navegación particular, como es el caso del filoguiado basado en visión.

Sin embargo, los modelos y sistemas desarrollados para tal fin se pueden reutilizar para plantear soluciones de navegación alternativas para el mismo prototipo o en cambio, ser empleado en otras plataformas.

## 2.2 Objetivos

El principal objetivo del presente proyecto es desarrollar un sistema de navegación viable para un prototipo de AGV. El desarrollo de dicho sistema de navegación puede dividirse en varios hitos. En cuanto al trabajo previo, relacionado con la elaboración del estado del arte:

- Revisión bibliográfica de las diferentes alternativas para el desarrollo de la navegación de un AGV.
- Revisión bibliográfica de los sistemas de navegación basados en visión artificial.
- Revisión bibliográfica de los modelos de redes neuronales convolucionales empleados en visión artificial para la clasificación de imágenes y detección de objetos.

En cuanto al desarrollo del sistema de navegación:

- Realizar un primer modelo del sistema de navegación empleando los modelos pre-entrenados presentes en la bibliografía.
- Reentrenamiento de los modelos pre-entrenados para adaptarlos al problema particular.
- Comparación de los resultados de los modelos empleados hasta el momento.
- Prueba de los modelos en el prototipo del robot.
- Desarrollo de un modelo final en vista de los resultados y la información obtenidos a partir de la revisión bibliográfica.

- Comparación entre los resultados del modelo final y los resultados obtenidos por los modelos previos
- Prueba del modelo final en prototipo de AGV

Además, en paralelo, se debe acondicionar el prototipo de AGV para emplearlo como plataforma de pruebas. Por tanto, resulta necesario:

- Diseñar e implementar el hardware para el control del prototipo de AGV.
- Programar el microcontrolador.
- Implementar comunicación entre el microcontrolador y el PC que procesa la visión.
- Realizar pruebas que demuestren que el sistema se ha implementado correctamente.

# 3 Estado del Arte

## 3.1 AGVs e Industria 4.0

Los AGV son robots móviles ampliamente utilizados en la industria para transportar mercancías entre dos puntos [1]. Los AGV forman una parte importante de los sistemas de transporte logístico en la industria. Se utilizan a gran escala, especialmente en Europa, desde hace más de una década [2].

En la actualidad, el mercado de los AGVs está creciendo rápidamente y es muy dinámico. Los AGVs se están integrando en sistemas de fabricación existentes, ya que proporcionan una serie de beneficios en materia de sostenibilidad económica, medioambiental y social [3], donde se incluye: Incremento en la productividad, reducción de los costes laborales, reducción de los costes energéticos y emisiones y una mejora de la seguridad.

La Industria 4.0, una iniciativa estratégica alemana, tiene como objetivo crear fábricas inteligentes en las que las tecnologías de fabricación se actualicen y transformen mediante sistemas ciberfísicos (CPS), el Internet de las cosas (IOT) y la computación en la nube [4]. Representa la cuarta revolución industrial en la fabricación. Sitúa la información en el centro, crea valor a partir de la información extraída y refinada de los datos. Permite la personalización masiva y una mayor integración horizontal y vertical [1].

Los sistemas AGV empleados actualmente y los sistemas actuales ofrecidos por los fabricantes mundiales funcionan casi todos bajo una forma de control centralizado: un controlador central controla toda la flota de AGVs.

Motivada por requisitos como la flexibilidad, la robustez y la escalabilidad, la tendencia actual de los sistemas AGV es la descentralización. La descentralización se define como la distribución de la inteligencia total de un sistema a sus componentes: cada dispositivo recibe una parte de la parte de la inteligencia total para poder funcionar de forma independiente. Algunos ejemplos de aplicación de la descentralización a un conjunto de AGVs son: el manejo del tráfico [5] y la gestión de los paros para recarga de baterías [6].

Para aprovechar el potencial de la Industria 4.0, los AGVs necesitan una arquitectura de control diferente que se apoye, junto a los Big Data, la interconectividad y la computación en la nube, en la descentralización. La inteligencia total de un sistema ya no estará centrada en una unidad de control, sino que todos los dispositivos tendrán su propia inteligencia creando datos para la recuperación de información independiente.

## 3.2 Navegación en los AGV

Un sistema de navegación completo para un AGV consta de múltiples tareas básicas para poder operar en entornos complejos. De Ryck et al. [1] distinguen 5 tareas básicas: Asignación de Tareas, Localización, Planificación de la trayectoria, Planificación del movimiento y Gestión del Vehículo. En la Figura 1 puede observarse un esquema que engloba las 5 tareas.

La primera tarea básica del AGV es la asignación de tareas. Bien se trate de un único AGV o de uno perteneciente a un enjambre, es necesario que un planificador le asigne cual es la tarea a realizar. En el primer caso, existen alternativas como anteponer las tareas más rápidas o

establecer un orden de prioridades. En cambio, para la segunda la forma más fácil de resolver el problema es asignar la tarea al AGV que está más cerca de la posición del objeto ordenado.

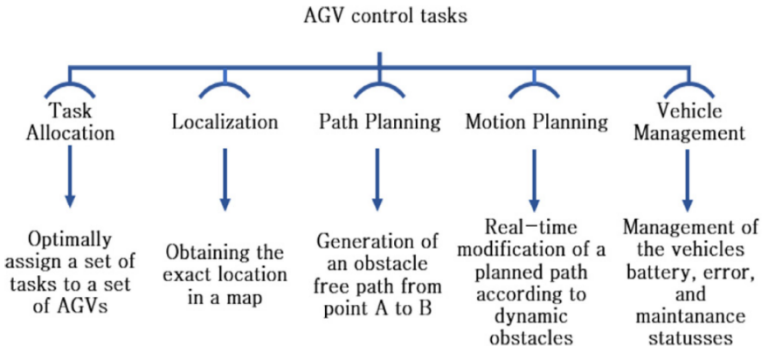



*Figura 1: Esquema de las 5 tareas básicas de un AGV [1].*

Se trata de un problema de optimización restringida de dificultad NP en el que el coste de la asignación total debe ser lo más bajo posible [7]. Si hay un número elevado de tareas y robots, el número de soluciones será enorme.

La siguiente tarea básica del AGV es la Planificación de la trayectoria. Se encarga de encontrar el camino más corto entre la posición actual del AGV y el punto de destino [8]. Utiliza una representación del entorno para buscar una secuencia de segmentos que permita alcanzar el objetivo lo más rápidamente posible. En la Figura 2 se muestra un esquema de la planificación de una trayectoria.

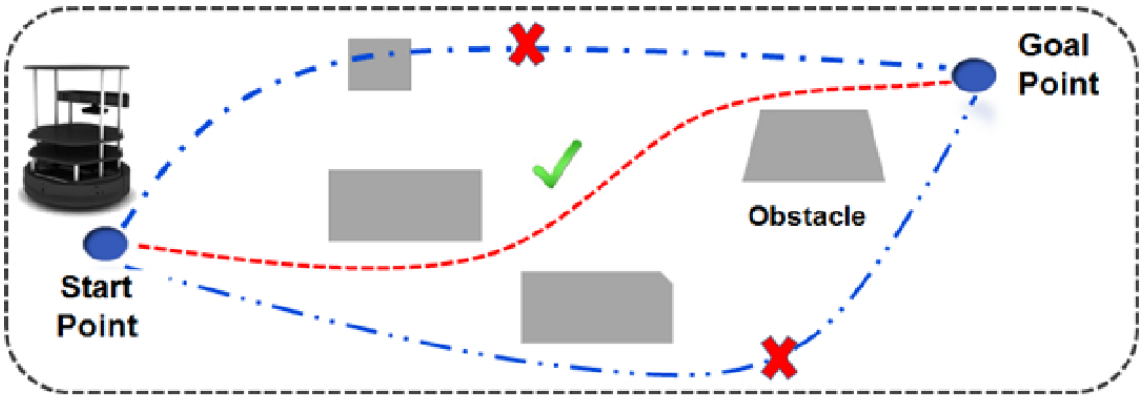



*Figura 2:* Ejemplo de la planificación de la trayectoria para un AGV

Interpretamos la planificación de la trayectoria como la tarea de planificación estática de un AGV, mientras que la planificación del movimiento que se va a tratar a continuación puede verse como planificación dinámica de la trayectoria [9]. En la planificación estática se calcula una trayectoria libre de colisiones utilizando información conocida.

Durante la ejecución de la ruta, el AGV puede enfrentarse a obstáculos que el planificador estático de la ruta desconoce. Estos obstáculos imprevistos pueden ser obstáculos dinámicos, personas y otros AGVs en movimiento. La colisión con estos elementos tiene que ser prevenida.

Además de las colisiones, también es necesario evitar bloqueos. Un bloqueo es una situación en la que un AGV ya no puede realizar ninguna acción. No puede avanzar ni retroceder. La modificación de una trayectoria estática predeterminada para evitar colisiones y bloqueos se denomina planificación del movimiento [10].

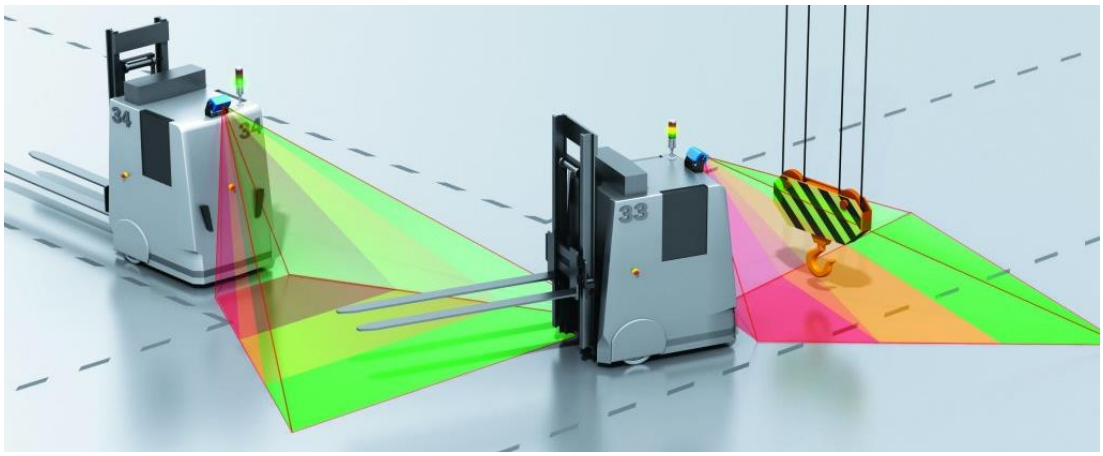



*Figura 3: Representación de la planificación del movimiento*

Paralelamente a estas tareas de los AGV, existe otra tarea central, la gestión de los vehículos, que controla y supervisa el estado de un AGV. La gestión del vehículo se encarga de comprobar la duración de la batería, los requisitos de mantenimiento y la gestión del estado de los errores. Una gestión de los vehículos aumenta la flexibilidad [11]. Es la tarea central más sencilla de los AGV en el sentido de que no requiere algoritmos o técnicas complejas. La identificación de errores causará restricciones en la posibilidad de realizar tareas y, por lo tanto, debe ser considerado en el nivel de asignación de tareas.

Finalmente, la última tarea por analizar es la localización del AGV, que al tratarse de un punto central del proyecto se estudia en una sección propia.

## 3.3 Localización de los AGV

En el control de los AGVs, la localización es una de las tareas claves a tener en cuenta. A diferencia de las demás tareas, ésta ya está descentralizada en su mayor parte: todo el equipo y el software de localización están abordo. La información sobre la ubicación en el mapa 2D del entorno puede comunicarse a un ordenador central o a los dispositivos vecinos para otros fines de control.

Las técnicas de localización pueden clasificarse en función del tipo de trayectoria como físicas o virtuales [12]. El primer grupo emplea un circuito físico para obtener una localización. Se denominan métodos de localización de la trayectoria física. El término "circuito" hace referencia al trazado de caminos e intersecciones por los que debe circular el AGV. Algunos ejemplos de técnicas que emplean este tipo de trayectoria son: Inductiva, Óptica y Cinta Magnética.

Otros métodos de localización no necesitan un circuito físicamente presente para localizarse. Estos métodos se denominan métodos de localización de trayectos virtuales. Emplean representaciones virtuales de los alrededores del AGV tanto en 2D como en 3D. Algunos

ejemplos serían: Puntos Magnéticos, Laser, GPS, Natural o de contorno y Guiado por visión. La Figura 4 muestra un esquema de la clasificación de los algoritmos de Localización.

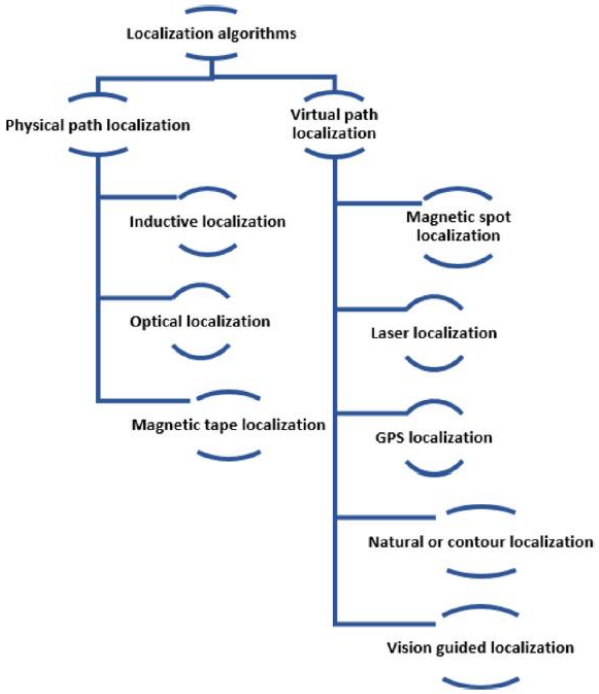



*Figura 4: Clasificación de los algoritmos de localización por [1].*

Teniendo en cuenta los requisitos de flexibilidad que exigen las nuevas tendencias en Industria 4.0, no es eficiente tener rutas físicas fijas que son difíciles de adaptar. Las futuras fábricas son dinámicas y pueden cambiar de configuración.

### 3.3.1 Métodos de localización de la trayectoria física

En la localización física, las trayectorias están presentes como guías físicas en el suelo y la localización se realiza mediante un sensor que detecta la guía [13]. Las trayectorias predefinidas se fijan en el suelo con cinta adhesiva o se incrustan en el suelo con un cable. El AGV no conoce realmente su posición dentro del mapa de la zona, sino que se limita a permanecer en la pista. Las marcas a lo largo de la pista pueden indicar al AGV si debe realizar una acción especial, como aumentar o disminuir la velocidad o girar en un ángulo determinado al encontrarse en una curva.

Localización Inductiva: Utilizado en la primera generación de AGVs. Se incrusta un cable en el suelo que al conducir una corriente eléctrica genera un flujo magnético [14]. El AGV lleva un sensor a bordo capaz de detectar dicho flujo. Se trata de una técnica probada, sobre todo en pasillos muy pequeños, para mantener la trayectoria con precisión. Sin embargo, no es un sistema flexible.

Localización Óptica: Se coloca una cinta de color o una línea pintada con alto contraste con el color del suelo [15]. El AGV tiene un sensor óptico a bordo. El principio de localización funciona de forma similar a la localización inductiva, compartiendo ventajas e inconvenientes. Una desventaja añadida es que la cinta puede ensuciarse o dañarse. Sin embargo, tiene un grado más elevado de flexibilidad y su coste es menor que la localización Inductiva.

Localización por Cinta magnética: La trayectoria se marca con una cinta magnética colocada en el suelo de la fábrica [16]. En el interior del vehículo, un sensor magnético es capaz de detectar dicho campo. El principio de localización es similar al de los métodos descritos anteriormente, compartiendo ventajas e inconvenientes.

En la Figura 5 se muestra un ejemplo de un AGV comercial guiado por cinta magnética. Los sistemas inductivos y ópticos muestran una presentación similar.

*Figura 5: AGV comercial guiado por cinta magnética*

### 3.3.2 Métodos de localización de la trayectoria virtuales

Con la localización virtual, las trayectorias están representadas virtualmente en un mapa local mantenido por el AGV o en el mapa global de la unidad central para el caso enjambres de AGVs con un sistema de control centralizado. Esto hace que sea fácilmente adaptable y ampliable. Sin embargo, la localización virtual es más compleja dado que el AGV necesita conocer su posición exacta en un mapa 2D. Esto contrasta con la localización de la trayectoria física, en la que el AGV sólo necesita conocer su posición respecto del circuito de referencia. Conociendo la ubicación en el mapa 2D, se pueden calcular las desviaciones de la trayectoria virtual.

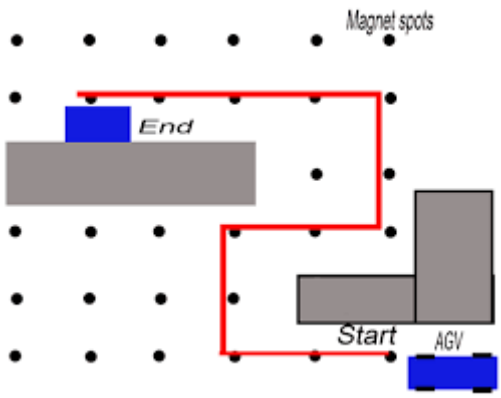



*Figura 6: Esquema de Localización por puntos Magnéticos [18].*

Localización por Puntos magnéticos: Se emplean imanes como balizas para definir una cuadrícula y por tanto un sistema de referencia (x, y) específicas del mapa de la zona [17]. Las balizas pueden ser imanes permanentes pasivos o transpondedores. En el interior del vehículo hay un sensor magnético capaz de detectar los puntos lo que permite al AGV determinar su posición absoluta en el mapa. La ubicación entre los puntos se recoge mediante posiciones relativas utilizando codificadores en las ruedas que calculan la distancia recorrida (odometría). Debido a la combinación de localización absoluta y relativa, este es un método robusto. Por otro lado, debido al tiempo requerido para instalarlo y modificarlo es poco flexible. La Figura 6 muestra un esquema de localización por puntos magnéticos.

Localización por Laser: Actualmente es el método más aceptado para la localización de AGVs. Consiste en un láser giratorio instalado sobre el vehículo. La localización se basa en la detección de múltiples puntos de referencia fijos, como tiras reflectantes, en el área de operación en coordenadas conocidas [19]. Las coordenadas de los reflectores se añaden al mapa global. Los rayos láser emitidos son reflejados y escaneados por el AGV, tras lo cual éste puede triangular su posición absoluta basándose en las coordenadas de los reflectores. Para poder navegar, deben ser visibles al menos tres puntos de referencia. Se trata de un método preciso, seguro y fiable. Las principales desventajas son su elevado precio y el esfuerzo que supone colocar todos los reflectores en la zona de la fábrica.

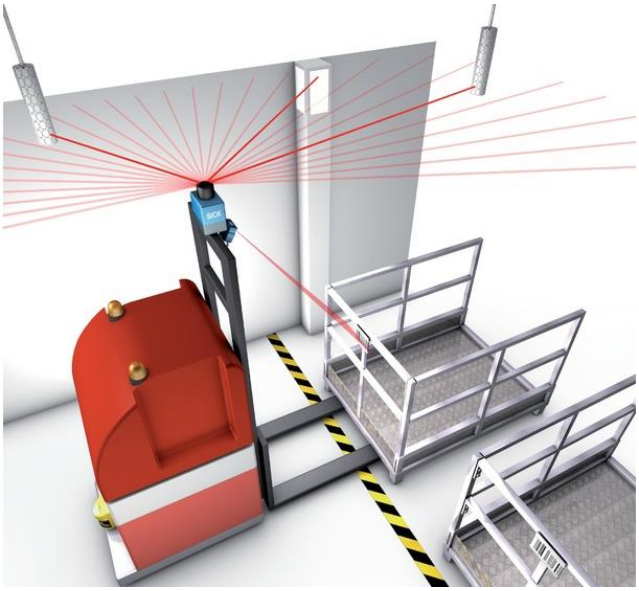
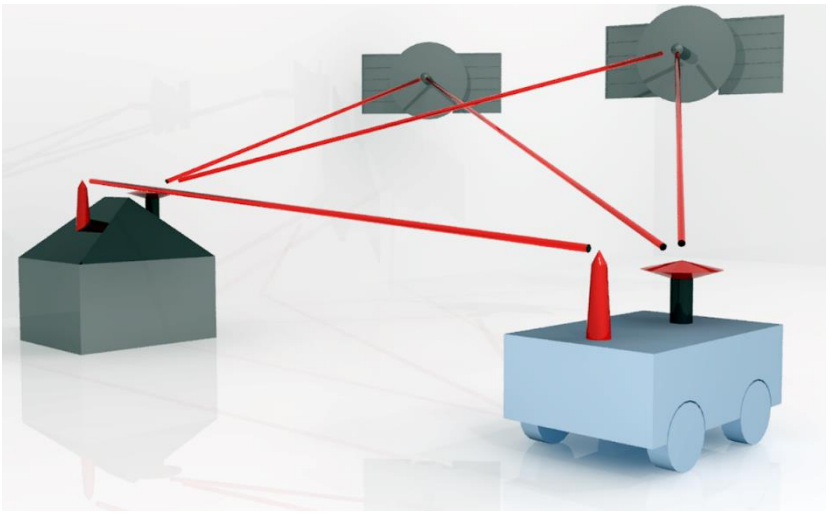


*Figura 7: Ejemplo de Localización por Laser*

Localización por GPS: Similar a la localización realizada por los sistemas de GPS integrados en los navegadores de a bordo de un vehículo [20]. Los satélites con posiciones conocidas en el mapa global emiten señales que son detectadas por el receptor GPS. Este receptor puede entonces medir la distancia a cada satélite. Esta información se utiliza para determinar la posición absoluta del receptor mediante la trilateración. Para poder navegar, deben ser visibles al menos cuatro satélites. En el ámbito Industrial es difícil de aplicar ya que requiere de una línea de visión clara hacia el cielo. Como alternativa, un radar de posicionamiento local (LPR) en la fábrica puede utilizarse en lugar de los satélites. La desventaja de este LPR es que tiene una

precisión de 10 cm, que dependiendo de la aplicación puede resultar insuficiente. Esta desventaja puede paliarse mediante la Fusión de sensores.



Localización Natural o por Contorno: Este tipo de localización utiliza un sensor de detección y alcance de luz (LiDAR) para escanear todo el entorno alrededor del vehículo. No se necesitan puntos de referencia fijos [21]. El AGV utiliza las características del entorno existente para navegar. Por ello se trata de un sistema de localización muy flexible, ya que no se necesita ninguna infraestructura adicional. Ahora bien, el método es menos preciso y robusto que otras alternativas debido a los reflejos y la deriva. Valiéndose del mapa escaneado del entorno, el vehículo puede hacer un mapa 2D de su entorno con todas las características visibles, como paredes y pilares. Al comparar este mapa local con el mapa de la fábrica, el robot puede inferir su posición en el mapa empleando la localización y el mapeo simultaneo (SLAM) [22]. mediante SLAM, el robot explora el área mientras utiliza escaneos láser para actualizar un mapa 2D local, lo que hace que el sistema sea mucho más flexible en entornos dinámicos. Las desventajas son que los sensores son caros y que algunos materiales transparentes no pueden ser detectados cuando se utiliza el láser. Otros sensores, como el sonar pueden utilizarse como alternativa.

*Figura 8: Sensor LiDAR del prototipo de AGV del departamento de sistemas de la EIV*

Localización por Visión: La localización guiada por visión es similar a la localización por contornos. En lugar de utilizar un LiDAR, se utiliza una cámara para obtener imágenes a partir de las cuales se pueden construir nubes de puntos 3D. Cada píxel de la cámara se convierte en un punto en el espacio 3D que se sitúa delante de la cámara. Esta nube 3D se compone de puntos

que representan características en el área vista por la cámara. Esta nube de puntos puede proyectarse en una nube de puntos 2D que transporte las características a las dos dimensiones. Al igual que en la localización natural, el robot también necesita un mapa inicial de su entorno que puede ser escaneado por una persona al explorar el área total utilizando SLAM. Una desventaja de este método es que las imágenes de la cámara son sensibles a las condiciones de luz que aparecen con frecuencia en los entornos de la vida real. El campo de la localización por visión ha sufrido un auge en los últimos años gracias al desarrollo de nuevas tecnologías en el campo del procesamiento de imagen.

### 3.3.3 Fusión de sensores

En la práctica, los métodos de localización anteriores no se aplican de forma independiente. Debido al ruido de los datos de los sensores y a la deriva, la incertidumbre en la posición medida es demasiado grande para que un AGV navegue adecuadamente cuando se utiliza un sólo tipo de localización. Por esta razón, los métodos de localización pueden combinarse entre sí o emplearse junto con filtros. La combinación de diferentes métodos de localización y filtrado se denomina fusión de sensores [23]. La fusión de sensores puede dividirse en fusión directa e indirecta. La fusión directa combina los datos de los sensores de diferentes sensores homogéneos o heterogéneos. La fusión indirecta combina los datos de los sensores con información previamente conocida del entorno y la entrada. La Figura 9 muestra un ejemplo de fusión de señores para mejorar la localización de un AGV.

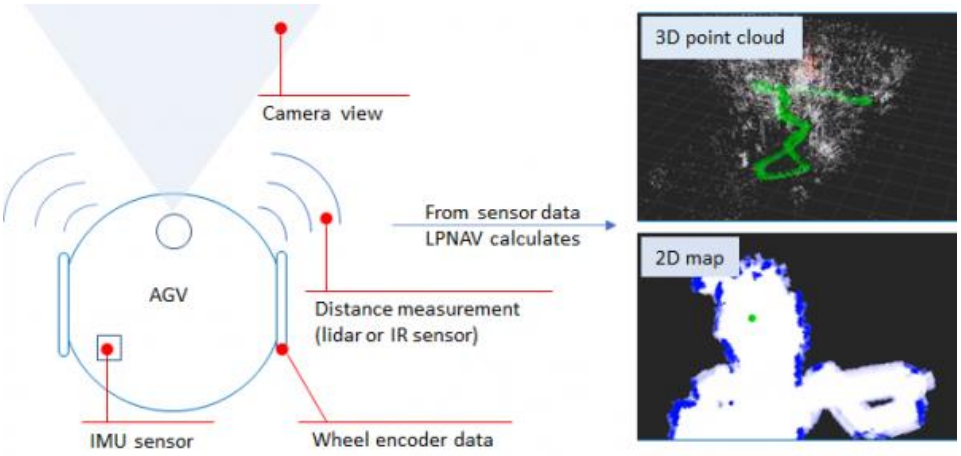



*Figura 9: Representación de AGV con fusión de sensores*

## 3.4 Inteligencia Artificial y Deep Learning

Debido a los recientes avances en Inteligencia Artificial (IA), Machine Learning (ML) y Deep Learning, varias aplicaciones de estas técnicas han ganado protagonismo y han pasado a primer plano en áreas Industriales punteras. Una de esas áreas es la conducción autónoma. La última década ha sido testigo del creciente interés de la investigación en la aplicación de la IA a la conducción de automóviles. Esta tendencia también se ha visto reflejada en los sistemas de Localización de los AGV, como se ha indicado en el punto anterior.

Sin embargo, para que estos coches se conviertan en una realidad funcional, deben estar dotados de percepción y cognición para enfrentarse a escenarios de alta presión en la vida real,

llegar a decisiones adecuadas y tomar las medidas más apropiadas y seguras en todo momento [24].

Las IA integradas, tanto en vehículos autónomos como en AGVs, son sistemas de reconocimiento visual que abarcan la clasificación de imágenes, la detección de objetos, la segmentación y la localización [25]. La detección de objetos está emergiendo como un subdominio de la visión por ordenador que se beneficia del Deep Learning, especialmente de las CNN.

### 3.4.1 Deep Learning

El Deep Learning es un modelo computacional de múltiples capas que se utiliza para la extracción de características y el aprendizaje de la representación en varios niveles de abstracción [25]. El Deep Learning es una rama del ML que extrae automáticamente características y patrones de los datos brutos y realiza predicciones o toma acciones basadas en alguna función de recompensa.

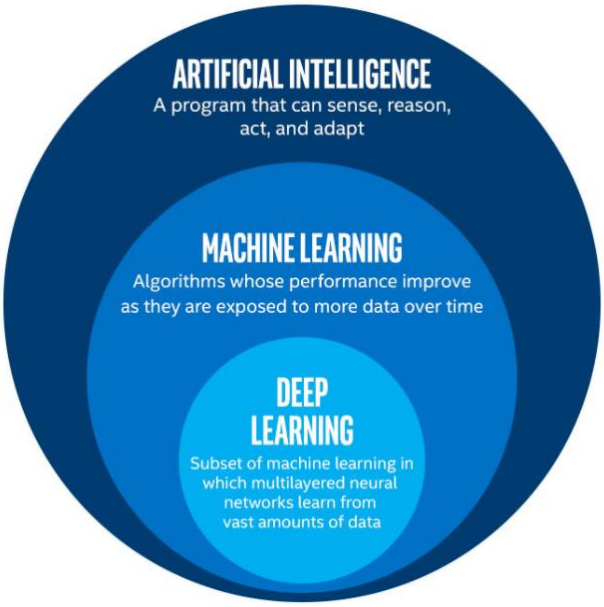



*Figura 10: Esquema de pertenencia. Inteligencia Artificial.*

Para lograr la detección de objetos, y la percepción de la escena, los vehículos autónomos deben percibir el entorno de una manera similar al ojo humano, lo que conduce a sistemas de IA cognitiva que pueden aprender, reaprender y realizar acciones. Para lograr una conducción de nivel humano desde el punto de vista de la visión por ordenador, los vehículos autónomos deben ser capaces de reconocer el entorno, discernir el movimiento de los objetos, los peatones y otros coches. Esta afirmación es válida también para los AGV industriales, ya que deberán de reconocer a trabajadores y obstáculos y otros AGVs presentes en la fábrica. La Figura 11 muestra un ejemplo de detección de objetos por parte de la cámara de un vehículo autónomo.

Los AGV utilizan Deep Learning para la planificación de la trayectoria, evitar obstáculos y procesar la información basada en la cámara para resolver problemas complejos de visión por computador [26]. Aunque los algoritmos de Deep Learning son capaces de percibir una gran cantidad de características, los costes que conlleva la anotación manual de los mapas restringen su aplicación a la conducción autónoma.

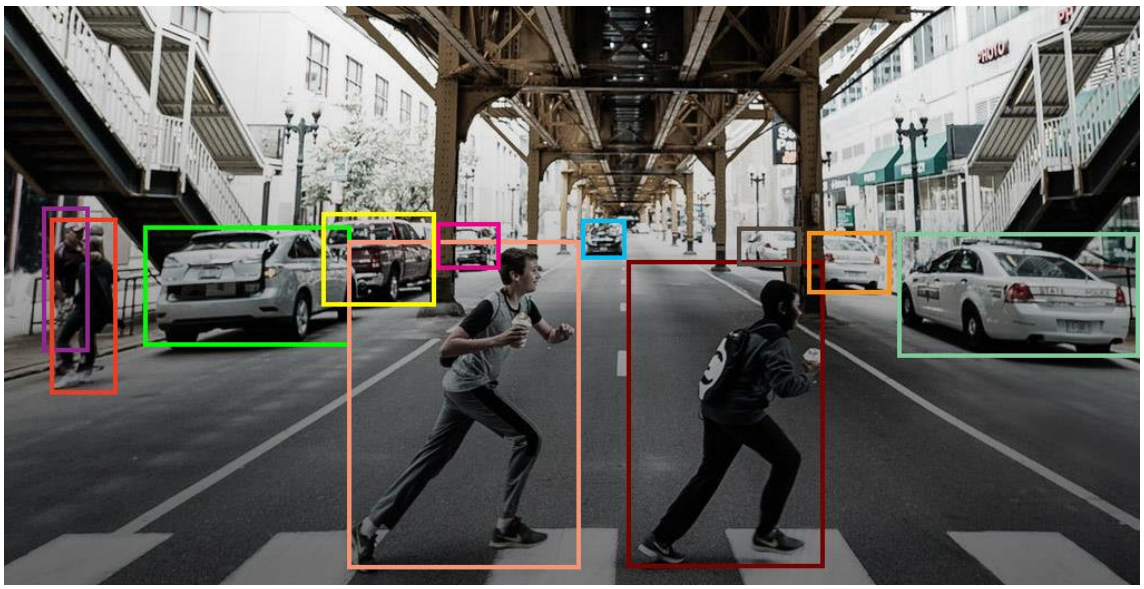


*Figura 11: Ejemplo de detección de Objetos en un vehículo Autónomo*

Algunos ejemplos del uso de Deep Learning en aplicaciones para la conducción autónoma son: El desarrollo de un algoritmo de navegación por Al-Qizwini et al. [27], un marco para la extracción, clasificación y finalización de marcas viales a partir de nubes de puntos tridimensionales por Wen et al. [28] y un sistema de interacción humano-AGV mediante gestos por Zhang et al. [29].

Dentro del Deep Learning existen diferentes arquitecturas aplicables a la conducción autónoma. Algunas arquitecturas de Deep Learning que se han aplicado para la detección de objetos en la conducción autónoma son: Las CNN, las redes neuronales recurrentes (RNN), “Deep belief Neural Network” (DBN) o los Autoencoders apilados (SAE).

RNN: Contienen conexiones cíclicas que las convierten en una herramienta más potente para modelar series temporales complejas que las redes neuronales convencionales. Se han aplicado para un seguimiento visual robusto y preciso en vehículos autónomos navegando escenarios con restricciones. Las RNN predicen la posición de un objeto en el siguiente fotograma basándose en la región de interés (ROI) y utiliza esa imagen como entrada del fotograma siguiente, asemejándose a un modelo de predicción para el seguimiento de objetos. Se ha empleado ampliamente para predecir las trayectorias de vehículos y peatones y evitar colisiones [30].

DBN: son un modelo gráfico generativo híbrido de múltiples capas utilizado para el aprendizaje de características robustas a partir de datos de alta dimensión y consiste en múltiples capas de variables estocásticas, latentes (ocultas) conectadas entre las neuronas de diferentes capas, pero no entre las unidades de cada capa [31]. Las máquinas de Boltzmann

restringidas (RBM), en las que cada capa se entrena por separado para producir una entrada esperada, son los bloques de construcción de la arquitectura DBN. Cada capa de RBM se comunica con las capas anteriores y siguientes para obtener precisión y eficiencia computacional. La Figura 12 muestra un esquema de la arquitectura DBN,

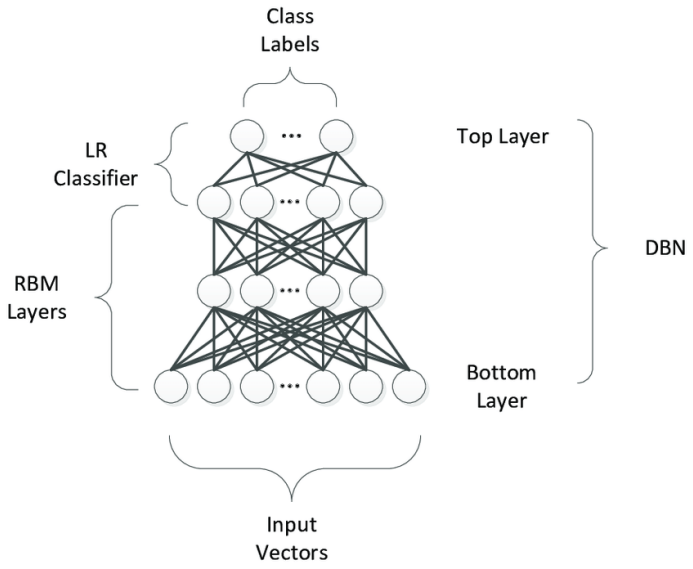



*Figura 12: Arquitectura de una red DBN [32]*

Autoencoders: Son una clase de extractores de características no supervisados que encuentran una transformación generalizada de la entrada y ayudan a un clasificador en una tarea supervisada [33]. Los SAE se utilizan en los sistemas de visión de los vehículos autónomos para visualizar datos de alta dimensión con el fin de encontrar clusters y crear métricas de similitud entre las muestras [34]. Las características aprendidas del autocodificador son tolerantes a los cambios en el espacio de entrada. La Figura 13 muestra una arquitectura encoder-decoder. Los encoder marcados como "$Encoder_1$" y "Encoder2" representan un ejemplo de Autoencoders apilados.

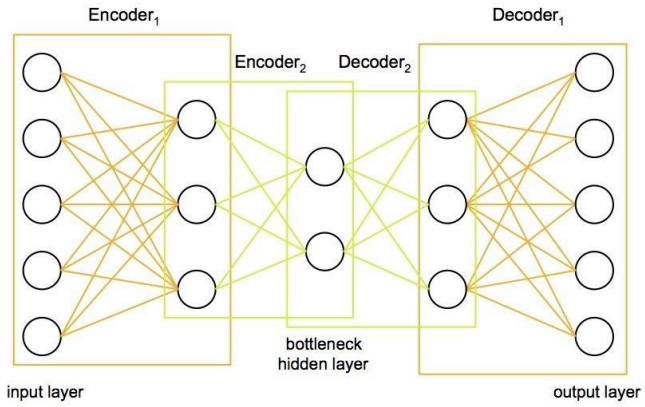



*Figura 13: Arquitectura Encoder-Decoder [35]*

## 3.5 Redes Neuronales Convolucionales (CNN)

Las CNN se han aplicado ampliamente a la clasificación de imágenes y a la visión por ordenador, y han obtenido tasas de clasificación del 100% en conjuntos de datos como ImageNet [36]. En la arquitectura de las CNN, las sucesivas capas de neuronas aprenden características progresivamente complejas de forma supervisada. La última capa representa las categorías de imágenes de salida. La Figura 14 muestra un ejemplo de arquitectura de red Convolucional (AlexNet).

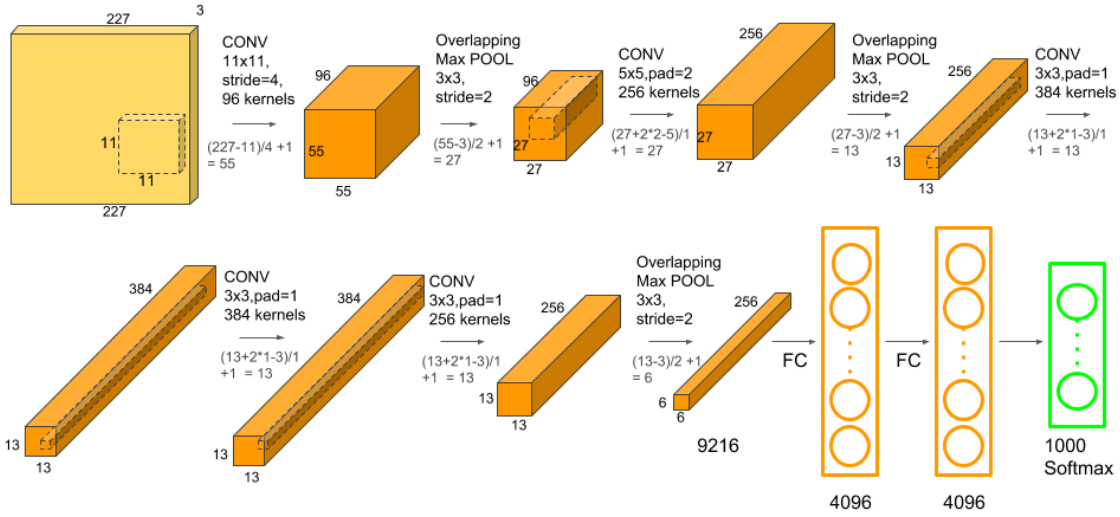



*Figura 14: Arquitectura de AlexNet [37]*

Las CNN no utilizan un módulo de extracción de características distinto ni un módulo de clasificación, es decir, eliminan la necesidad de describir y extraer características manualmente (Figura 15). Extrae las características de los datos brutos basándose en los valores de los píxeles, lo que conduce a las categorías finales de los objetos (Figura 16).

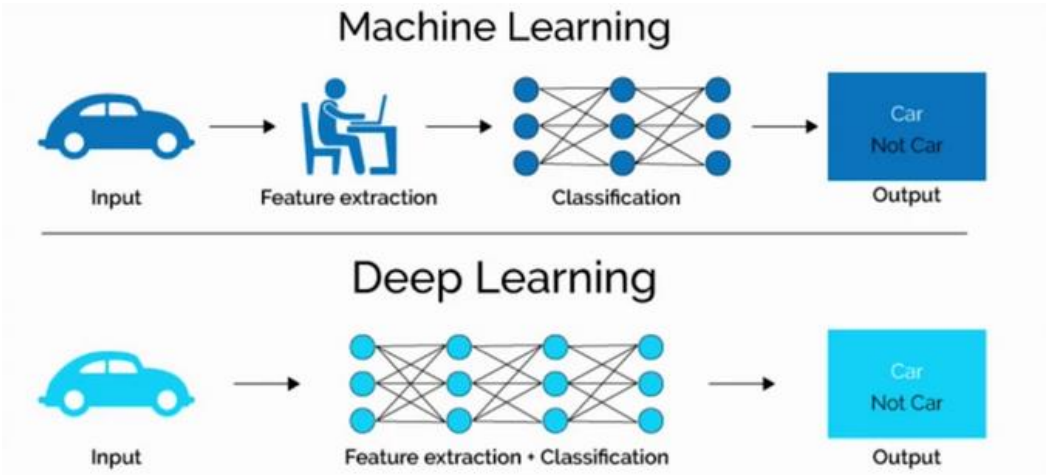



*Figura 15: Comparación Machine Learning vs. Deep Learning*

Una CNN consta principalmente de tres partes: la capa de convolución, la capa de "Pooling" y la capa "Fully Conected" [38]. La salida de esta última puede considerarse una característica aprendida, que se utilizará en la clasificación o detección cuando se conecte a un clasificador al final.

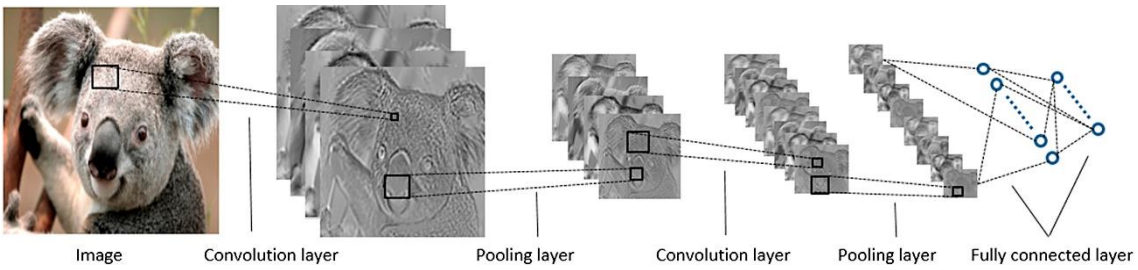



*Figura 16: Extracción de características en una CNN [39]*

### 3.5.1 Capas principales en una Red Neuronal convolucional

Convolución: Una capa de convolución genera nuevas imágenes llamadas mapas de características. El mapa de características acentúa las características únicas de la imagen original. Una capa de convolución no funciona de la misma forma que las capas de una red neuronal convencional. La Figura 17 muestra un ejemplo de aplicación de un operador de convolución con un filtro de tamaño 3x3.

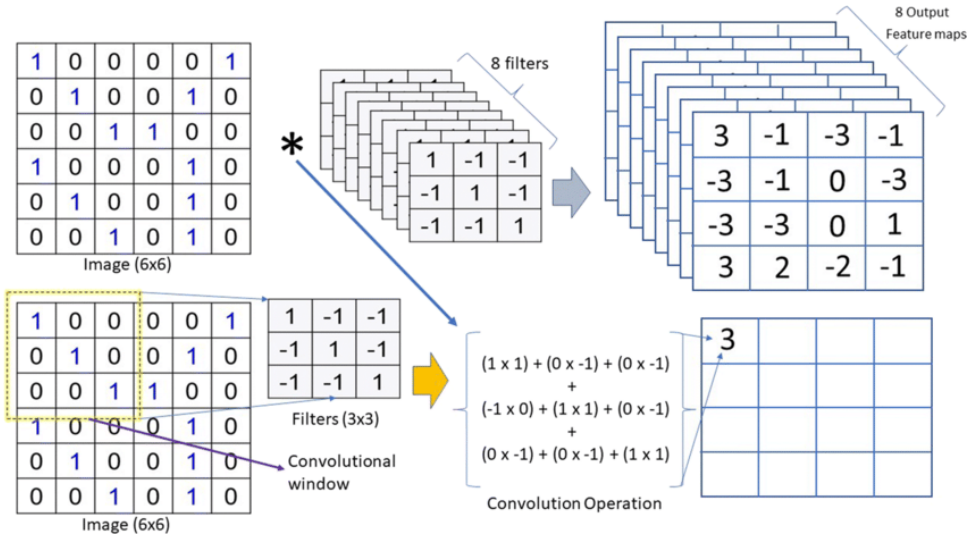



*Figura 17: Ejemplo de aplicación de la capa de Convolución [40]*

La capa de convolución no emplea pesos sinápticos ni una suma ponderada. En su lugar, contiene filtros que convierten las imágenes. A estos filtros se les denominan filtros de convolución. Los mapas de características se generan al aplicar los filtros de convolución a la imagen original. La Figura 18 contiene un ejemplo de la variedad de mapas de características que se pueden obtener al aplicar filtros con distintos valores.

Pooling: La capa de Pooling reduce el tamaño de la imagen, ya que combina los píxeles vecinos de una determinada zona de la imagen en un único valor representativo [41]. Los píxeles vecinos se suelen seleccionar de la matriz cuadrada, y el número de píxeles que se combinan difiere de un problema a otro. El valor representativo suele ser la media o el máximo de los píxeles seleccionados. En la Figura 19 se muestra un ejemplo de aplicación de la capa de "Pooling" con un operador de dimensión 2x2. El tamaño del operador puede variar entre problemas. Se puede observar la diferencia al seleccionar el valor medio o el máximo como valor representativo.

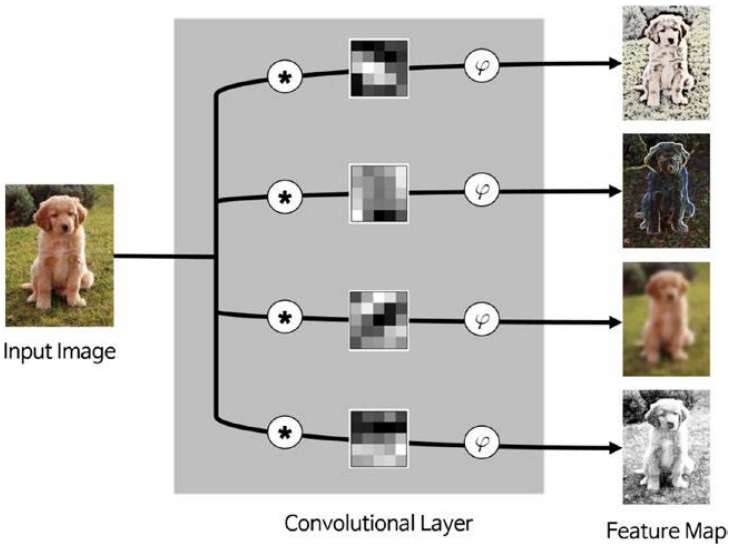



*Figura 18: Mapas de características [41]*

Fully Conected: Esta capa se comporta de manera idéntica que una red tradicional, como las redes "Multi Layer Perceptron" (MLP). La salida de la capa totalmente conectada puede considerarse la característica aprendida. La operación de clasificación puede completarse conectando un clasificador, como una capa tipo Softmax. Para completar el entrenamiento de la CNN es necesario conectar un clasificador. Dependiendo de la elección del clasificador, la sensibilidad de la última capa será diferente. Sin embargo, los resultados de la capa de convolución y capa de Pooling permanecerán constantes.

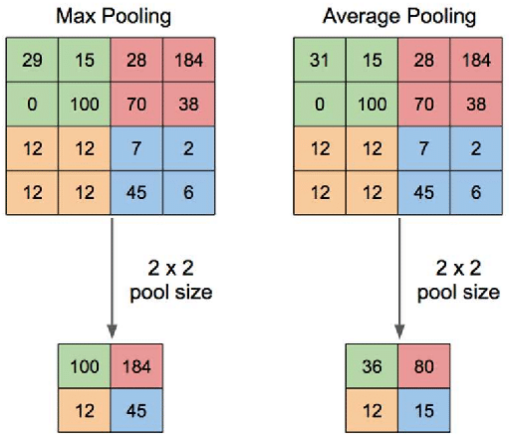



*Figura 19: Ejemplo de aplicación de la capa "Pooling" [42]*

### 3.5.2 Tipos de clasificadores en una Red Neuronal Convolucional

Las redes Neuronales convolucionales pueden realizar diferentes tipos de clasificación de imagen en función del tipo da capa de salida escogida. Como se ha comentado en la sección anterior, a parte de los tres tipos principales de capa una CNN tiene, como elemento final, un clasificador.

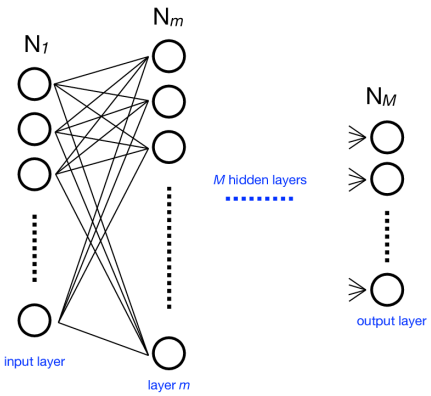



*Figura 20: Ejemplo de capa Fully Conected [43]*

Este clasificador es el que traduce las características extraídas en información procesable. Dentro de los modelos de CNN entrenados de forma supervisada destacan tres tipos de clasificador: Clasificación de Imágenes, Detección de objetos y Segmentación semántica.

Clasificación de imágenes: Consiste en asignar una etiqueta o serie de etiquetas a la imagen. En ML existen pequeños ejemplos de clasificación de imágenes, que ante una imagen de entrada son capaces de asociarla con una etiqueta. Un ejemplo típico es la clasificación de las flores de Iris [44] que permite distinguir tres subtipos de Imagen o la clasificación de razas de perros y gatos [45] que permite distinguir hasta 37 especies diferentes. Cada etiqueta es denominada, comúnmente, como clase.

El uso del Deep Learning en este tipo de clasificaciones ha permitido incrementar tanto el número de clases como la capacidad de clasificación de los modelos. Modelos como AlexNet [36], GoogLeNet [46] o VGG [47] son capaces de devolver un vector de salida en el que se recoge la probabilidad de que cierta clase esté presente en una Imagen. La Figura 21 muestra la arquitectura del modelo VGG. Se puede observar que su salida es vector de dimensión 1000 lo que indica que es capaz de identificar mil clases distintas dentro de una imagen.

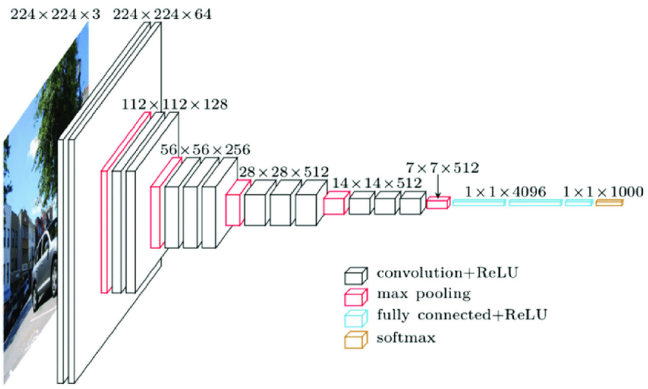
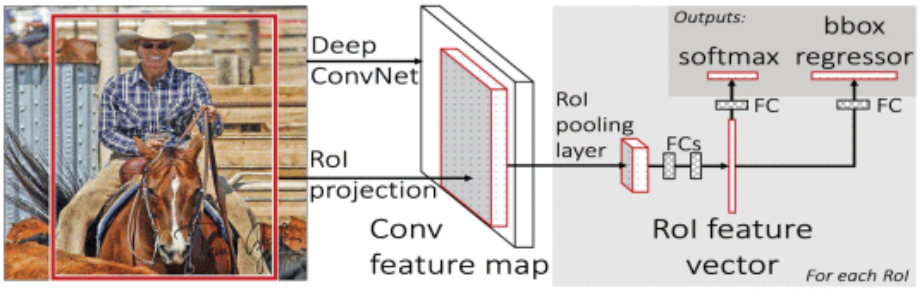
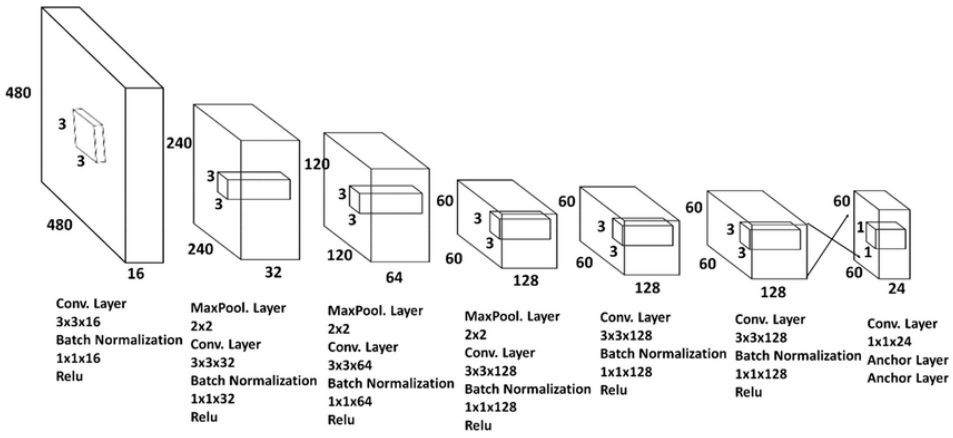



*Figura 21: Arquitectura de la red VGG [48]*

Detección de objetos: Su objetivo es localizar y clasificar los objetos existentes en una imagen cualquiera y etiquetarlos con una "Bounding Box" rectangular [49]. Se pueden distinguir dos tipos de detección de objetos. Uno de ellos sigue el procedimiento tradicional de detección de objetos, generando primero propuestas de regiones y clasificando después cada propuesta en diferentes categorías de objetos. El otro considera la detección de objetos como un problema de regresión o de clasificación, adoptando un marco unificado para obtener directamente los resultados finales (categorías y localizaciones). El principal exponente de la primera clase es el modelo R-FCN [50] mientras que en el segundo encontraríamos el modelo YoLoV2 [51].




*a)*




*b)*

*Figura 22: Esquema de arquitecturas de detección de objetos: a) R-FCN [49]. b) YoLoV2 [52]*

Segmentación Semántica: Tiene como objetivo asignar una etiqueta a cada píxel de una imagen [53]. Los modelos CNN han permitido un notable progreso en las tareas de segmentación semántica por píxeles debido a la estructura jerárquica de sus características y a la capacidad de entrenar dichos modelos de principio a fin. En la segmentación semántica, el mapa de etiquetas de salida tiene el mismo tamaño que la imagen de entrada. Como puede observarse en la Figura 23, este mapa de salida marca de un color diferente cada una de las clases detectadas. En este caso detecta la posición de una persona en la imagen.

Es el método de detección más preciso en cuanto a clasificar correctamente la posición. Sin embargo, esta precisión conlleva un alto coste computacional. La profundidad de estos modelos requiere de un gran número de imágenes para el correcto entrenamiento de los modelos. Esto unido a la dificultad de encontrar ejemplos etiquetados previamente para la realización de entrenamientos supervisados representa el principal inconveniente en su desarrollo.

La mayoría de los sistemas de segmentación semántica más avanzados, son modificaciones de modelos de clasificación y detección de imagen y presentan tres componentes clave: Una “Fully Convolutional Network” (FCN) que sustituye las capas “Fully Conected” para realizar un aprendizaje e inferencia eficientes de extremo a extremo. Campos aleatorios condicionales (CRF), para capturar las dependencias locales y de largo alcance dentro de una imagen para refinar el mapa de predicción. Y finalmente, una convolución dilatada, que se utiliza para aumentar la resolución de los mapas de características intermedias con el fin de generar predicciones más precisas manteniendo el mismo coste computacional.

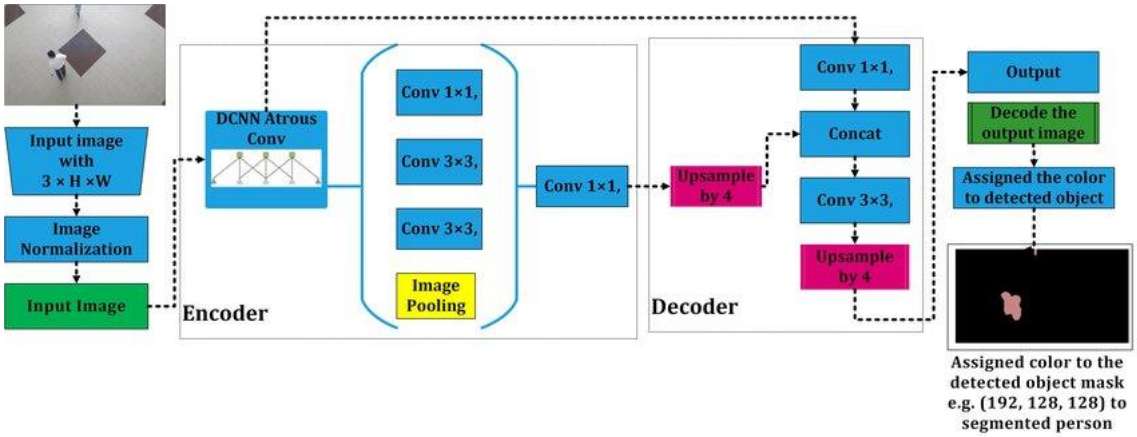



*Figura 23: Arquitectura de la red DeepLabV3*

## 3.6 Detección de Líneas

En la localización de Vehículos Autónomos, el empleo de las señales de circulación está muy extendido. La detección de las líneas presentes en la carretera permite corregir la ruta del vehículo y mantenerse dentro de la trayectoria [54].

La detección de estas líneas se ha beneficiado de las nuevas técnicas en Inteligencia artificial y Deep Learning desarrolladas en los puntos anteriores. Esto ha permitido un reconocimiento más preciso y rápido de dichas marcas viales y mejorado la conducción.

Este método de localización ha sido extrapolado a AGVs industriales [55]. La detección de líneas es un caso particular de la detección y clasificación de objetos cuyos métodos han sido abordados en el apartado: 3.5.2 Tipos de clasificadores en una Red Neuronal Convolucional.

Como especifican Tang et al. [56] “Como aplicación específica, existen muchas estrategias para la detección de carriles. Desde la perspectiva de cómo definir la tarea de detección de carriles, puede resumirse en tres las categorías existentes: 1) métodos basados en la clasificación, que combinan algunos datos previos para determinar la posición de los carriles; 2) método basado en la detección de objetos. Mediante el etiquetado por “Bounding Box”, para cada segmento de carril, los carriles pueden detectarse por regresión de coordenadas; 3) método basado en la segmentación semántica. Los carriles y los píxeles del fondo se etiquetan como clases diferentes. Y los resultados de la detección pueden obtenerse en forma de clasificación a nivel de píxel.”

Desde el punto de vista de la estructura del modelo, puede resumirse en los dos tipos siguientes: Modelo de tarea única y modelo multitarea. El primero sólo se considera la detección de carriles, y no se incluyen otras señales de tráfico. El modelo multitarea, que combina la detección de

carriles con otras tareas, como la detección de zonas transitables, el reconocimiento de marcas viales, el tipo de carretera o la clasificación del tipo de carril. En la práctica, muchas estructuras e ideas disponibles pueden extraerse de las CNN existentes, como VGG, ResNet y la arquitectura de extremo a extremo de FCN, etc.

### 3.6.1 Detección de líneas mediante clasificación de imágenes

La aplicación de la clasificación de imágenes generalmente tiene como objetivo discriminar qué objeto contiene la entrada. A todas luces, esta forma no puede obtener la ubicación de los carriles. Es necesario utilizar una conversión entre la clasificación y la detección de carriles.

DeepLanes [57] es un método basado en esta idea. Se trata de un modelo de CNN específicamente entrenado para detectar líneas. Las imágenes de entrada de dicho entrenamiento tienen una resolución de 240 × 360. Fueron obtenidas por cámaras montadas lateralmente en el vehículo y orientadas hacia el suelo. Para obtener una distribución de probabilidad de las posiciones de los carriles, se aplica la función softmax a la salida de la última capa totalmente conectada dando un vector con 317 salidas: 316 clases posibles para las posiciones del carril y una clase para la ausencia de carril. La Figura 24 muestra el tipo de salida que ofrece la red DeepLanes. Los puntos amarillos se corresponden con las salidas activadas de la red. A partir de ellas se puede reconstruir el trazado de la línea detectada.

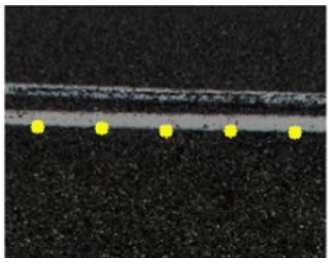


*Figura 24: Ejemplo de la salida de la Red DeepLanes [57]*

### 3.6.2 Detección de líneas mediante detección de objetos

Para la detección de objetos, se persiguen dos objetivos principales: predecir categorías de objetos en una imagen y determinar la posición de los objetos detectados.

Un ejemplo de ello es el modelo EElane [58]. Emplea un modelo de regresión para la detección de líneas y vehículos. La regresión de carriles predice un vector de seis dimensiones. Las primeras cuatro dimensiones indican dos puntos finales de un segmento de línea local, y las dos dimensiones restantes indican la profundidad de los puntos finales con respecto a la cámara.

Otro ejemplo es la “Spatial Temporal Lane detection Network” (STLNet) [59]. Se compone de tres pasos: preprocesamiento, clasificación y regresión basada en CNN, y ajuste de carriles. En STLNet, se emplean CNN para clasificar el tipo de límite y hacer una regresión de la posición de los límites del carril. La principal característica de STLNet es que explora las características de las restricciones temporales.

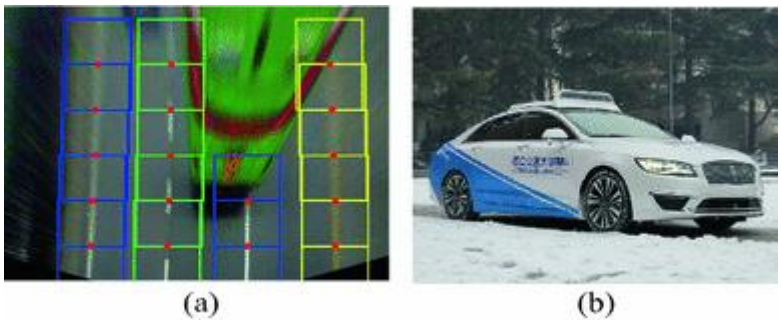



*Figura 25: Ejemplo de la salida de la Red STLNet [59]*

### 3.6.3 Detección de líneas mediante segmentación semántica

En el caso de la detección de líneas, la técnica más sencilla es resolver el problema mediante la segmentación de dos clases. Esta es la estrategia adoptada por el modelo LaneNet [60]. La segmentación de la imagen tiene dos clases "Línea" y "Entorno". La Figura 26 muestra un ejemplo de este tipo de segmentación binaria a una imagen tomada en una autovía. La máscara mostrada representa con un "1" los pixeles clasificados como línea y con un "0" los clasificados como entorno.

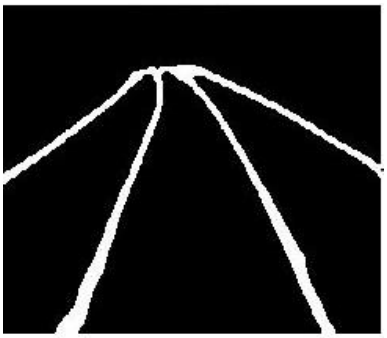


*Figura 26: Mascara de la segmentación de líneas en una autopista*

Una de las características por las que LaneNet destaca frente a otros métodos de segmentación es su capacidad de identificar líneas por separado. LaneNet, a diferencia de otros métodos que solo son capaces de distinguir que pixeles pertenecen a línea y cuales, a entorno, aplica un método de clusterización en su entrenamiento para distinguir entre líneas.

Este proceso de clusterización se realiza en paralelo al entrenamiento de la segmentación de líneas lo que dota al modelo de mayor robustez. La Figura 27 muestra el ejemplo de la detección de líneas realizado por el modelo LaneNet.

*Figura 27: Ejemplo de la salida de la Red LaneNet [61]*

# 4 Desarrollo de la solución

## 4.1 Formulación del Problema

En consecuencia, con los objetivos marcados en la sección 2.2 del proyecto, se busca diseñar un sistema de navegación y localización, basado en CNN, para el prototipo de AGV del laboratorio de proyectos del Ingeniería de Sistemas y Automática.

Este sistema de navegación y localización debe de permitir al AGV recorrer los pasillos adyacentes al laboratorio evitando colisiones con las paredes o posibles obstáculos y siguiendo la trayectoria fijada. La Figura 28 muestra un ejemplo del pasillo a recorrer por AGV. La trayectoria queda definida por medio de una cinta amarilla.

*Figura 28: Pasillo de la EIV*

Para desarrollar el sistema de localización, será necesario emplear alguno de los sistemas descritos en el apartado 3.3 Localización de los AGV, de tal forma que el prototipo sea capaz de determinar su situación en el propio pasillo, con respecto a la línea amarilla que traza la trayectoria.

A partir de esta información el sistema de navegación debe de ser capaz de traducir la posición que ocupa el AGV respecto de la trayectoria en unas consignas de velocidad. Estas consignas de velocidad se corresponden con las velocidades lineales de las ruedas del AGV.

Así mismo, el prototipo presente en el laboratorio es una carcasa vacía, por lo que debe de desarrollarse un sistema hardware, basado en un microcontrolador, capaz de comunicarse con la unidad de procesamiento de imagen para recibir los valores de las consignas. Este sistema también debe de poder controlar los motores presentes en el AGV. Finalmente será necesario programar el microcontrolador para cumplir con las especificaciones citadas.

La solución propuesta se validará mediante una serie de pruebas que medirán la capacidad del sistema de localización para detectar la trayectoria delimitada por la línea. El seguimiento de esta por parte del AGV y los tiempos de ejecución.

## 4.2 Herramientas y Equipos Empleados

En esta sección se realiza una descripción de todas las herramientas y equipos que han sido necesarias para desarrollar el proyecto. Cubre tanto elementos de hardware o herramientas, como software empleado para la programación.

### 4.2.1 Prototipo de AGV

Se emplea un modelo Pioneer 3 fabricado por Mobile-Robot. Pioneer es una familia de robots móviles, tanto de dos como de cuatro ruedas motrices, que incluye el Pioneer 1 y el Pioneer AT, el Pioneer 2-DX y los más recientes robots móviles Pioneer 3-DX y -AT. Estas pequeñas plataformas de investigación y desarrollo comparten una arquitectura y un software base comunes con el resto de las plataformas de Mobilerobots.


*Figura 29: AGV Pioneer 3 original*

Este AGV portaba su propio hardware de control con un microprocesador de 32KB. Dado que este software viene asociado a un sistema cerrado de desarrollo se opta por retirar el software original y sustituirlo por unos microcontroladores de Arduino. De esta manera se consigue un control directo sobre el software de control de las ruedas motrices.

Únicamente se mantienen los motores de cada una de las ruedas. En la Figura 30 se puede observar el modelo del laboratorio una vez retirado el hardware original.

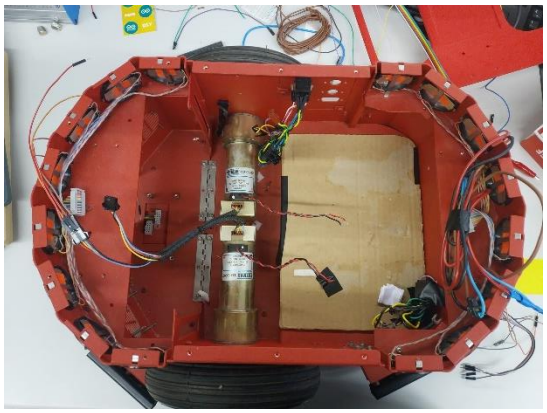


*Figura 30: Imagen del Pioneer 3-DX del laboratorio desprovisto del hardware original*

### 4.2.2 Arduinos MEGA y UNO

Como se ha comentado en la sección previa ha sido necesario sustituir el hardware original del AGV por otro del que se tenga conocimiento sobre el software en ejecución. Este microcontrolador debe cumplir con las siguientes características:

- Capacidad de trabajar de forma embebida.
- Capacidad de comunicación vía puerto serie con otros equipos.
- Capacidad de controlar los motores del AGV directamente o a través de otros circuitos compatibles.

Se decide emplear dos microcontroladores de Arduino. En concreto los modelos Uno y MEGA. El Arduino MEGA se emplea en el proceso de configuración y testeo de la comunicación por puerto serie debido a que dispone de más de un puerto serie, a diferencia del modelo UNO.

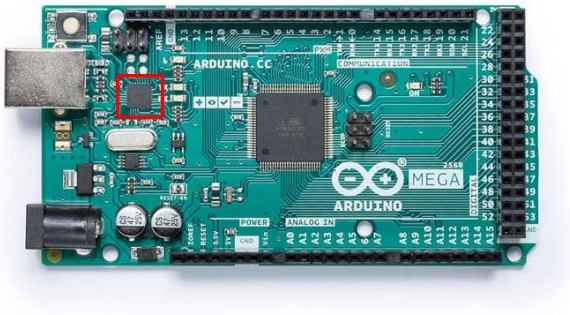



*Figura 31: Arduino MEGA*

Emplear varios puertos a la vez permite realizar un "ECO" de las consignas enviadas desde el PC que procesa la imagen y visualizarlas en pantalla mediante la herramienta Putty.

El modelo UNO se coloca dentro del hardware de control del AGV, ya que una vez testeada la comunicación, un único puerto serie es suficiente.


*Figura 32: Arduino UNO*

### 4.2.3 Cámara Web

Para implementar la localización por visión es imprescindible añadir un periférico de visión al prototipo de AGV. Se utiliza una webcam comercial debido a su compatibilidad.

Esta webcam permite tomar imágenes y video en varias resoluciones. Se toma la resolución más pequeña 360x640. La webcam permite conexión directa con el PC mediante el puerto USB.

Matlab, el software de procesamiento de imagen elegido es capaz de detectar y manejar la webcam mediante funciones de alto nivel. Esto permite añadir la toma de imágenes al algoritmo de localización.

*Figura 33: Webcam acoplada sobre el AGV*

### 4.2.4 Putty

PuTTY es un emulador de terminal gratuito y de código abierto, una consola en serie y una aplicación de transferencia de archivos en red. Soporta varios protocolos de red, incluyendo SCP, SSH, Telnet, rlogin, y la conexión de socket crudo. También puede conectarse a un puerto serie.

Es la herramienta empleada para leer los "ECOS" reenviados por el Arduino MEGA. Esto permite comprobar que las tramas que contienen las consignas de velocidad se han enviado correctamente.

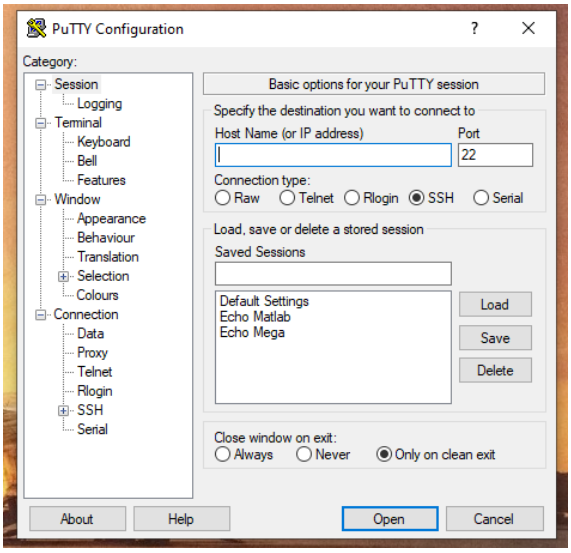



*Figura 34: Interfaz de Putty*

### 4.2.5 IDE Arduino

El Arduino IDE contiene un editor de texto para escribir código, un área de mensajes, una consola de texto, una barra de herramientas con botones para funciones comunes y una serie de menús. Se conecta al hardware Arduino y Genuino para cargar programas y comunicarse con ellos.

Es la herramienta utilizada para la programación de los microcontroladores empleados en el hardware de control y proceso de testeo del proyecto. En la sección Código de control de Motores en Arduino se detalla el software desarrollado en la plataforma.

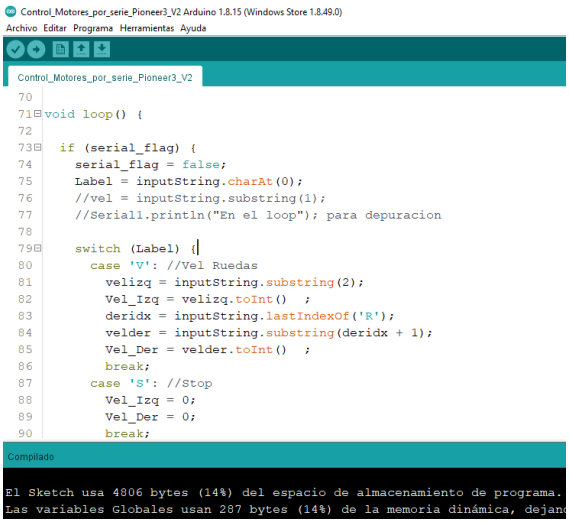



*Figura 35: Extracto del código de control desarrollado*

### 4.2.6 Matlab

Matlab es una plataforma de programación diseñada específicamente para que ingenieros y científicos analicen y diseñen sistemas y productos que transformen nuestro mundo. El corazón de Matlab es el lenguaje Matlab, un lenguaje basado en matrices que permite una expresión más natural de las matemáticas computacionales.

Esta ha sido la principal plataforma de desarrollo y testeo en el proyecto. El diseño y entrenamiento de las CNN se ha realizado empleando la toolbox de Deep Learning. Esta toolbox permite el uso de GPUs para el entrenamiento de las redes de forma nativa lo que reduce considerablemente el tiempo de entrenamiento. En la sección Código para el entrenamiento de Redes Neuronales Convolucionales en Matlab se detalla el procedimiento y el código empleados.

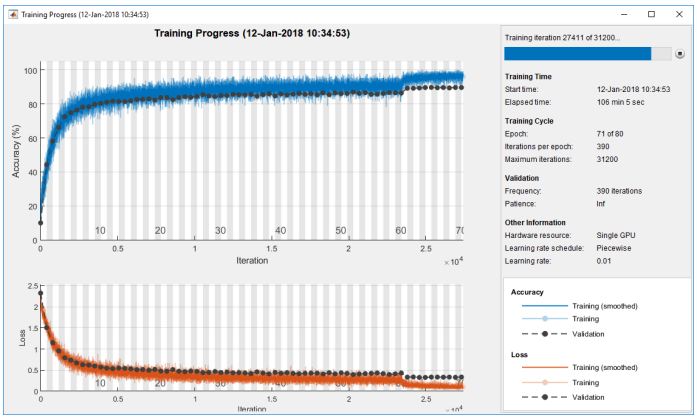



*Figura 36: Entrenamiento de una CNN mediante la toolbox de Deep Learning de Matlab*

Por otro lado, también ha sido la plataforma en la que se ha ejecutado el software de navegación. Matlab permite la programación de scripts que se ejecutan de forma cíclica. Es capaz de reconocer periféricos como webcams y controlarlos para tomar instantáneas. Esto permite al programa desarrollado conectarse a la cámara situada en el AGV, tomar una instantánea del pasillo y procesarla mediante el algoritmo de Navegación. Finalmente, también es capaz de enviar información vía puerto serie.

Estas cualidades unido al amplio conocimiento de la plataforma, la convierten en la más adecuada para el desarrollo del software de navegación. En la sección Código para el procesamiento de Imagen y Navegación en Matlab se detalla el programa de navegación desarrollado.

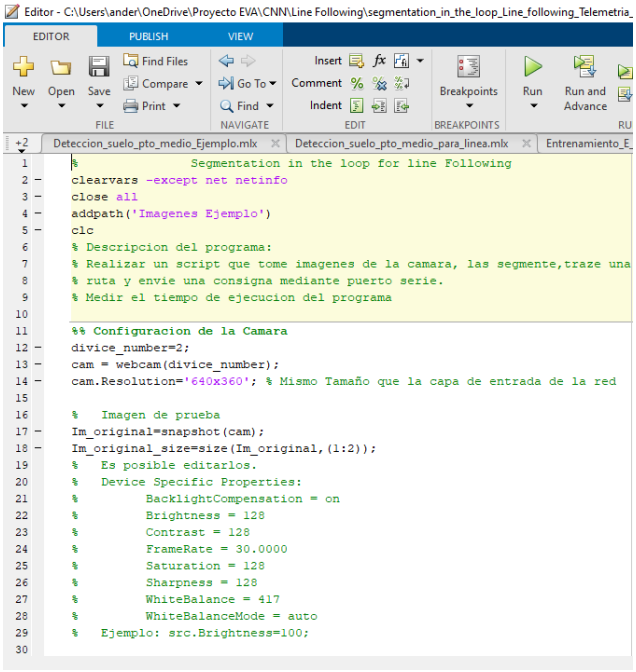



*Figura 37: Extracto del código de navegación*

### 4.2.7 Servidor remoto

La capacidad de cómputo de las GPUs determina la velocidad con la que un equipo es capaz de ejecutar y entrenar modelos de CNN. Esta diferencia de tiempo de capacidad de cómputo entre GPUs y CPUs y entre distintos modelos de GPUs pueden llegar a reducir los tiempos de entrenamiento se semanas a horas. Esto es debido a la gran cantidad de imágenes necesarias para realizar un correcto entrenamiento de los modelos de CNN.

Debido a ello, en el proceso de entrenamiento de los modelos de CNN desarrollados en el proyecto se ha empleado un servidor perteneciente al departamento que cuenta con dos GPUs de alta capacidad:

- NVIDIA Quadro RTX 6000
- NVIDIA Quadro P5000

Dicho servidor corre un sistema operativo Windows 10 que cuenta con la misma versión de Matlab empleada para la programación del sistema de navegación. La Figura 38 muestra un resumen de las características del resto de componentes del servidor.

Edición de Windows

Windows 10 Enterprise LTSC

© 2018 Microsoft Corporation. Todos los derechos reservados.

Sistema

| | |
|---|---|
| Procesador: | Intel(R) Xeon(R) Gold 5120 CPU @ 2.20GHz 2.19 GHz (2 procesadores) |
| Memoria instalada (RAM): | 96,0 GB (94,6 GB utilizable) |
| Tipo de sistema: | Sistema operativo de 64 bits, procesador x64 |
| Lápiz y entrada táctil: | La entrada táctil o manuscrita no está disponible para esta pantalla |

*Figura 38: Características del Servidor Remoto*

### 4.2.8 Bases de Datos de Imágenes

El entrenamiento de una CNN para la segmentación semántica requiere de una base de datos de gran tamaño, con el suficiente número de imágenes para garantizar la precisión del modelo entrenado.

En el desarrollo del proyecto se emplean modelos pre-entrenados para la segmentación semántica. Dichas arquitecturas no están específicamente diseñadas para la detección de líneas, por lo que es necesario realizar un reentrenamiento que permita obtener un modelo adecuado.

En la literatura existen varias bases de datos de imágenes especializadas en la detección de carriles. El inconveniente de estos "Dataset" es que están enfocadas a las líneas de carretera, para permitir la localización de vehículos autónomos. No existe una base de datos especifica de carriles para AGVs de interior.

Algunas de estas bases de datos revisadas durante el proyecto son: CULane [62] y LLAMAS [63].

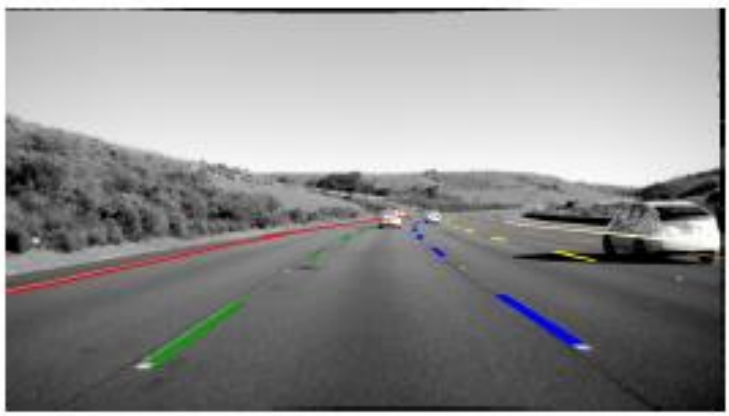


*Figura 39: Ejemplo de imagen perteneciente a la base de datos LLAMAS*

### 4.2.9 Circuitos de ensayo

Para poder comprobar que el sistema de localización y navegación es adecuado se han construido dos pequeños circuitos en el laboratorio de proyectos y en el pasillo adyacente.

El circuito marca, mediante cinta amarilla, la trayectoria que debe seguir el AGV. El primero de los circuitos es un tramo recto. El segundo es una trayectoria curva. La Figura 40 muestra el recorrido del segundo circuito en el interior del laboratorio.

*Figura 40: Circuito con trayectoria Curva*

## 4.3 Solución Propuesta

En consecuencia, según lo establecido en el punto 4.1 Formulación del Problema y para satisfacer los objetivos marcados en el unto 2.2 Objetivos, se propone un sistema de navegación y localización para el vehículo consistente en:

- Un hardware para el prototipo de AGV basado en un Arduino UNO como microcontrolador.
- Una Cámara Web instalada en el prototipo de AGV.
- Un PC portátil para el procesamiento de la imagen y la ejecución del algoritmo de navegación.
- Un programa para el microcontrolador que reciba consignas de velocidad por puerto serie. Las consignas se envían desde el PC que procesa las imágenes.
- Un programa para que el microcontrolador traslade las consignas, recibidas desde el PC, a los puertos PWM del Arduino. Estos a su vez, están conectados al driver L298N encargado de manejar los motores del AGV.
- Una trama para la comunicación serie.
- Un subprograma en Matlab capaz de enviar las consignas de velocidad por el puerto serie siguiendo la estructura de la trama definida previamente.
- Un programa de Matlab que permita el entrenamiento de modelos de CNN.
- Un modelo de CNN que segmente la imagen tomada por la cámara y destaque la línea sobre el entorno.
- Un algoritmo de localización, implementado en Matlab, que mide el error respecto de la trayectoria definida por la línea amarilla y calcule la acción de control.

La estructura final del AGV consiste en el armazón del robot Pioneer 3 al que se ha acoplado la webcam en su parte delantera. En el interior se coloca la nueva configuración de hardware basada en la placa de Arduino UNO. Finalmente, el PC portátil, que cuenta con una GPU adecuada para la ejecución de modelos CNN se sitúa sobre la tapa del AGV. La Figura 41 muestra la estructura final del AGV.

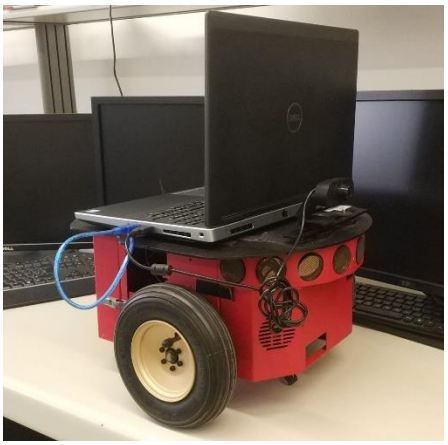


*Figura 41: Estructura final del prototipo de AGV*

Integrando los programas en las plataformas de Matlab y Arduino, enumerados previamente, se compone el algoritmo de navegación y localización para AGV, que le permitirá seguir dentro de la trayectoria. El diagrama de flujo de la Figura 42, muestra el ciclo de ejecución del algoritmo.

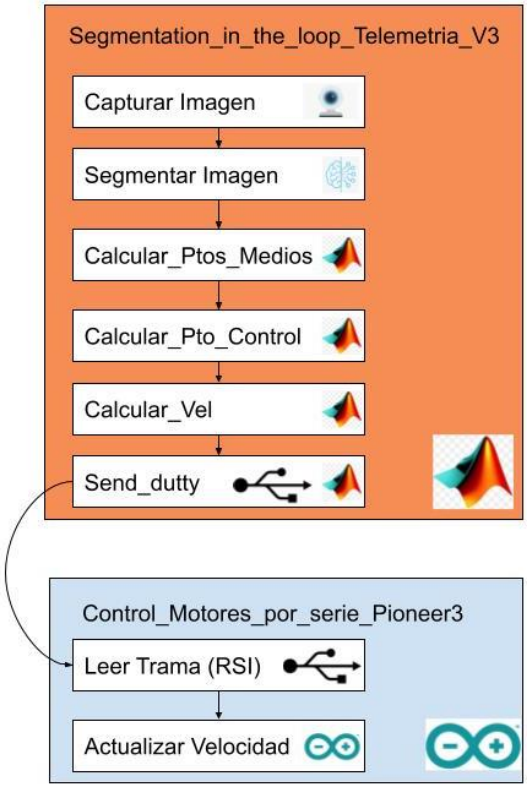



*Figura 42: Diagrama de flujo del algoritmo de localización y navegación*

El ciclo de ejecución sigue los siguientes pasos:

1. El programa principal de Matlab toma una captura del frontal del AGV mediante la webcam.
2. La imagen obtenida es procesada por la CNN para obtener una máscara que destaque la línea marcada en el suelo.
3. Se identifica la trayectoria a recorrer por el AGV a partir de la máscara.
4. Se mide el error entre la trayectoria detectada en el punto anterior y la trayectoria actual del robot.
5. Se calcula la acción de control en función del error.
6. Se envía la acción de control desde MATLAB al Arduino.
7. El Arduino actualiza la velocidad de los motores.

En el siguiente punto se detalla la metodología seguida para desarrollar los programas de Matlab y Arduino. El procedimiento seguido para el entrenamiento de las redes, y el diseño del nuevo hardware del AGV

## 4.4 Metodología

En esta sección se describen los siguientes procedimientos:

- Entrenamiento modelos de CNN.
- Definición de la trayectoria a partir de la segmentación.
- Modelo matemático para el seguimiento de la trayectoria.
- Definición de la trama para la comunicación serie.
- Especificación del hardware instalado en el AGV.
- Programación del microcontrolador de control

Cada uno de estos procedimientos se analiza en un apartado independiente.

### 4.4.1 Entrenamiento de las CNN

Para entrenar, un modelo CNN, que realice tareas de segmentación semántica es necesario: Definir la arquitectura del modelo, establecer el conjunto de imágenes con sus etiquetas para el entrenamiento supervisado e implementar un programa que ejecute el entrenamiento.

En el análisis del del estado del arte, en el punto 3.5.2, se ha comentado que algunos de los modelos de CNN, pre-entrenados para tareas de segmentación semántica, son modificaciones de modelos CNN diseñados para clasificación de imágenes.

Este aprovechamiento del aprendizaje de un modelo preexistente, para la generación de un nuevo modelo con un propósito diferente se denomina “Transfer Learning”. En el caso de las CNN para segmentación semántica, el “Transfer Learning” aprovecha el conocimiento adquirido por el modelo pre-entrenado en las capas iniciales. Por otro lado, reestructura y reentrena las capas finales, realizando el cambio del tipo de clasificador.

La biblioteca de ejemplos de Matlab muestra un procedimiento para entrenar una red DeeplabV3 [64], un tipo CNN diseñada para segmentación semántica. En este ejemplo se realiza un “Transfer Learning” desde una red Res-Net18, de la que se extraen los pesos de las capas

iniciales. Este ejemplo emplea el conjunto de datos de CamVid [65]. Este conjunto de datos es una colección de imágenes que contienen vistas a nivel de calle obtenidas mientras se conduce. El conjunto de datos proporciona etiquetas a nivel de píxel para 32 clases semánticas que incluyen coche, peatón y carretera.

El modelo de DeeplabV3 desarrollado en dicho ejemplo permite segmentar imágenes de la carretera distinguiendo 11 clases diferentes. En la Figura 43 se muestra un ejemplo de imagen segmentada por la red DeeplabV3. La figura muestra la máscara obtenida tras la segmentación superpuesta sobre la imagen original permitiendo comprobar, mediante el código de colores de la derecha, el resultado de la clasificación a nivel de pixel.


*Figura 43: Ejemplo de imagen segmentada por la red DeeplabV3*

La red CNN desarrollada en el proyecto sigue el mismo procedimiento que el ejemplo citado. Para adaptar el ejemplo al problema particular de la segmentación de líneas se cambia el conjunto de datos de imágenes por el de CULane [62].

Empleando el código presente en los Anexos: Entrenamiento de la Red CNN, se obtiene una red CNN basada en la arquitectura DeepLabV3, que permite segmentar las imágenes de los circuitos de pruebas y obtener una máscara en la que se destaque la posición de la línea amarilla.

Este proceso tiene una etapa previa que consiste en redimensionar las imágenes del conjunto de datos. Para ello en primer lugar se debe establecer cuál será el tamaño de entrada de las imágenes de la CNN. El tamaño de las imágenes repercute notablemente en la velocidad de procesamiento del modelo y aumenta el tiempo de entrenamiento.

Un aspecto a tener en cuenta es la resolución ofrecida por la cámara web empleada para las tareas de visión. La cámara seleccionada (punto 4.2.3) ofrece varias resoluciones. Definir el tamaño de la capa de entrada de la CNN diferente a uno de los ofrecidos por la cámara, obligaría a realizar un redimensionamiento de la instantánea capturada en cada ciclo de ejecución, incrementando el coste computacional del algoritmo.

Por tanto, se define el tamaño de entrada de la red igual a la menor resolución ofrecida por la cámara: 360x640 (en pixeles). Esto, a su vez, obliga a redimensionar la base de datos CULane al

nuevo tamaño. SU dimensión original es de 590x1640. Empleando el código "Redimensionamiento de bases de datos" descrito en los anexos se generan dos nuevas carpetas que contienen las imágenes originales y las etiquetas.

Con el conjunto de imágenes y las etiquetas separados en sendas carpetas, el siguiente paso en el proceso de entrenamiento es la generación de las estructuras "datastore" en Matlab. Estas estructuras aceleran el acceso a la memoria en el proceso de entrenamiento ya que permiten a Matlab acceder a las imágenes leyéndolas directamente del disco sin tener que cargarlas en el "workspace".

imageDS.Files

| | 1 |
|---|---|
| 1 | C:\Users\ander\Documents\CuLane datasheet\driver_161_90frame\06030819_0755.MP4\00000.jpg |
| 2 | C:\Users\ander\Documents\CuLane datasheet\driver_161_90frame\06030819_0755.MP4\00090.jpg |
| 3 | C:\Users\ander\Documents\CuLane datasheet\driver_161_90frame\06030819_0755.MP4\00180.jpg |
| 4 | C:\Users\ander\Documents\CuLane datasheet\driver_161_90frame\06030819_0755.MP4\00270.jpg |
| 5 | C:\Users\ander\Documents\CuLane datasheet\driver_161_90frame\06030819_0755.MP4\00360.jpg |
| 6 | C:\Users\ander\Documents\CuLane datasheet\driver_161_90frame\06030819_0755.MP4\00450.jpg |
| 7 | C:\Users\ander\Documents\CuLane datasheet\driver_161_90frame\06030819_0755.MP4\00540.jpg |
| 8 | C:\Users\ander\Documents\CuLane datasheet\driver_161_90frame\06030819_0755.MP4\00630.jpg |
| 9 | C:\Users\ander\Documents\CuLane datasheet\driver_161_90frame\06030819_0755.MP4\00720.jpg |
| 10 | C:\Users\ander\Documents\CuLane datasheet\driver_161_90frame\06030819_0755.MP4\00810.jpg |
| 11 | C:\Users\ander\Documents\CuLane datasheet\driver_161_90frame\06030819_0755.MP4\00900.jpg |
| 12 | C:\Users\ander\Documents\CuLane datasheet\driver_161_90frame\06030819_0755.MP4\00990.jpg |
| 13 | C:\Users\ander\Documents\CuLane datasheet\driver_161_90frame\06030819_0755.MP4\01080.jpg |
| 14 | C:\Users\ander\Documents\CuLane datasheet\driver_161_90frame\06030819_0755.MP4\01170.jpg |
| 15 | C:\Users\ander\Documents\CuLane datasheet\driver_161_90frame\06030819_0755.MP4\01260.jpg |

*Figura 44: Vista de la estructura de datos que almacena las imágenes*

Cabe destacar, que, al generar la estructura correspondiente a las etiquetas, no es suficiente con añadir las máscaras que indican a que clase pertenece cada pixel. Se debe añadir además el nombre asociado a cada clase y su código numérico. En el caso que nos ocupa, los nombres de las clases son: "línea" y "No línea". Para el identificador numérico, dado que se trata de un caso de segmentación de solo dos clases, se emplean el "1" y el "0" respectivamente. La Figura 45 muestra el ejemplo de una etiqueta perteneciente al conjunto de datos CULane.

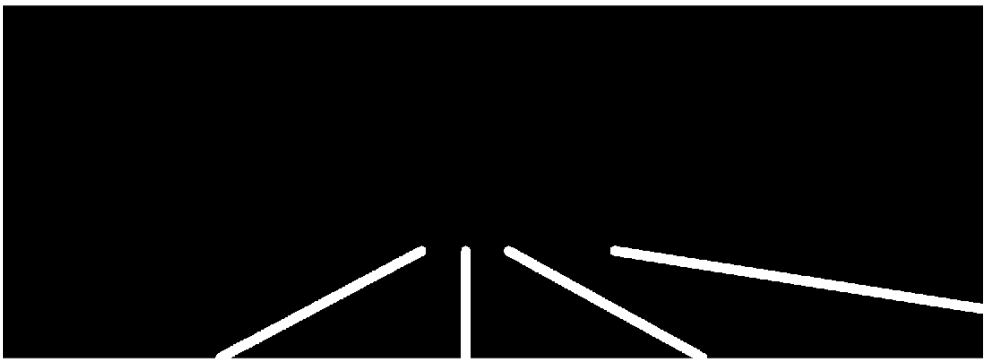


*Figura 45: Etiqueta perteneciente al conjunto de datos CULane*

Otro paso a tener en cuenta es el análisis estadístico del conjunto de datos. En el ejemplo de segmentación semántica de Matlab se hace un recuento de total de pixeles y cuál de las 11 clases del rediseño del banco de imágenes CamVid pertenece. En dicho análisis se puede observar como la frecuencia de aparición de las clases no es homogénea y algunas son más predominantes. La Figura 46 muestra la frecuencia relativa de las 11 clases del rediseño del banco de imágenes CamVid.

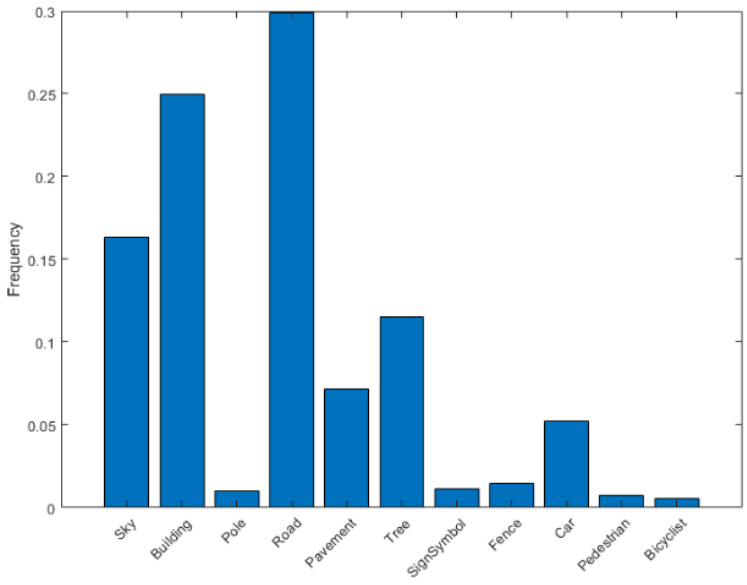



*Figura 46: Frecuencia relativa de las clases presentes en CamVid*

Este análisis es necesario ya que las frecuencias relativas calculadas son empleadas en el proceso de "transfer Learning". Tras definir la arquitectura de la red DeepLabV3, y antes de realizar el reentrenamiento, se añaden los inversos de las frecuencias relativas a la capa final de clasificación. El objetivo es ponderar la detección de las clases, de tal forma que se reduzca el error al clasificar las clases menos predominantes.

Chougule et al. [66] definen el concepto de objeto clase débil ("weak class object"). "Un objeto de clase débil, en el contexto de la segmentación semántica, ocupa relativamente pocos píxeles en la escena. A su vez, la precisión en la detección de dichos objetos es relativamente baja." También exponen que es un problema de especial importancia en el caso de la detección de carriles, ya que estos suelen representar un porcentaje pequeño de pixeles respecto del total de la imagen. Por tanto, es imprescindible aplicar técnicas, como la ponderación de la capa de clasificación, para mejorar la detección de los objetos de clase débil.

Finalmente, el último paso antes de comenzar con el entrenamiento es la separación de la base de datos en grupos de entrenamiento, test y validación. Al igual que en otras aplicaciones de IA este paso es crucial para evitar el sobre entrenamiento.

Una vez entrenado el modelo, es necesario comprobar su precisión a la hora de clasificar los circuitos descritos en el punto 4.2.9. Para ello, en 5.1 Diseño de las pruebas, se especifican las simulaciones realizadas con imágenes tomadas desde el AGV en parada para testear el desempeño del modelo.

Observaciones al modelo obtenido: Tras el reentrenamiento, el modelo de DeepLabV3 fue testado con un pequeño banco de imágenes tomadas de los circuitos. Los resultados obtenidos no eran satisfactorios. El modelo entrenado no era capaz de destacar las líneas con la precisión requerida. Para remediar esta situación, se optó por realizar un reentrenamiento empleando imágenes de las líneas amarillas de los circuitos, aplicando técnicas de "data augmentation".

En el siguiente punto se aborda como obtener una base de datos y sus etiquetas para segmentación semántica a partir de un conjunto de datos empleando la herramienta "Image labeler" de Matlab.

### 4.4.2 Base de datos propia mediante Image Labeler

Ante la imposibilidad de encontrar un conjunto de datos de imágenes, enfocado en líneas para AGVs industriales, y la falta de precisión del primer modelo desarrollado, entrenado mediante transfer Learning y el conjunto de datos CamVid, se acude a la herramienta Image Labeler de Matlab.

Esta herramienta permite realizar etiquetados manuales para tareas de clasificación de imágenes. Entre sus opciones permite etiquetar las imágenes empleando "Bounding Box", líneas y pixel. Para mantener la lógica establecida a lo largo del proyecto se usa la última opción en el etiquetado de una pequeña muestra de imágenes, que contiene imágenes de las líneas amarillas tomadas desde el AGV.

Este conjunto de datos posee 65 imágenes, como la mostrada en la parte izquierda de la Figura 47. Empleando la herramienta de Image Labeler, se definen dos clases "Línea" y "No Línea". Una vez definidas la herramienta permite marcar las partes de la imagen correspondientes a cada una de las clases, como puede observarse en la propia figura. Mediante el color rojo se marcan los pixeles pertenecientes a la clase "Línea". En azul a la clase "No Línea".

*Figura 47: Ejemplo de etiquetado empleando Image Labeler*

Una vez finalizado el proceso se genera una carpeta con las máscaras que se emplearán en el reentrenamiento de la red. Estas máscaras, se guardan junto a una estructura de datos que permite asociar las etiquetas categóricas "Línea" y "No Línea" con su correspondiente valor numérico. La Figura 48 muestra la representación de los identificadores numéricos definidos en dicha estructura. Estas máscaras y su información asociada se emplean en el proceso de entrenamiento descrito en el Anexo: Entrenamiento de la Red CNN.

El modelo obtenido mediante esta metodología es capaz de segmentar las imágenes de los circuitos de ensayo con la suficiente precisión. En la sección 5.2 Análisis de Resultados, se especifica la precisión de dicho segmentado.

*Figura 48: Mascara obtenida mediante el Image Labeler*

Sin embargo, este modelo también es descartado, debido a los requisitos de tiempo de ejecución, marcados en el punto 5.1 Diseño de las pruebas. A continuación, se especifica la metodología seguida para rediseñar la arquitectura del modelo, empleando estrategias para redes CNN de ejecución en tiempo real.

### 4.4.3 Rediseño de la arquitectura para tiempo real (ED-CNN)

Si bien, el modelo del punto anterior es preciso a la hora de detectar las líneas del recorrido de pruebas, a la hora de probar el sistema completo del AGV, se observa que el seguimiento de la trayectoria no es satisfactorio. En el punto 5.2 Análisis de Resultados se ofrecen más detalles.

El origen del problema es la velocidad de ejecución del modelo. En [67] Paszke et al. aseguran que una CNN aplicada a tareas de tiempo real debe de tener una velocidad de ejecución que le permita procesar, al menos, 10 imágenes por segundo (fps). El modelo presentado el punto anterior no es capaz de alcanzar dicha tasa de fps.

Por tanto, tras realizar una nueva revisión bibliográfica se opta por seguir la arquitectura propuesta por Chougule et al. [66]. En este trabajo proponen un nuevo modelo de CNN basada en una arquitectura encoder-decoder, de la que se extrae su nombre ED-CNN.

La segmentación de los límites de los carriles puede considerarse un caso especial de segmentación semántica de dos clases. Es una tarea más sencilla que la segmentación multiclase. Por lo tanto, el uso de una red diseñada para una segmentación multiclase (como SegNet, ENet y FCN) para resolver el problema de la segmentación de carriles, produce una infrautilización de la arquitectura base.

ED-CNN es una CNN cuya arquitectura consta de 10 capas de convolución dividas en los bloques de “encoding” y “decoding”. En la primera parte se sitúan bloques de normalización entre las capas de convolución. La Figura 49 muestra un esquema de la arquitectura de la red ED-CNN.

En el bloque de “encoding” de esta red aprovecha los pesos de otras redes como E-Net [67] y FCN-Alexnet [68] mientras que el bloque de “decoding” es completamente reentrenado.

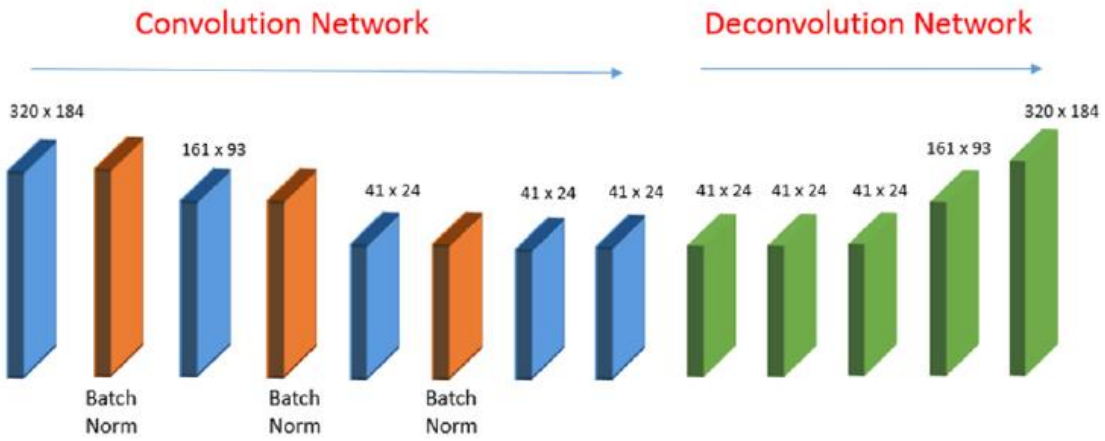



*Figura 49: Arquitectura de la red ED-CNN [66].*

El modelo obtenido cumple con los requisitos de precisión y tiempos de ejecución establecidos en el apartado de Diseño de las pruebas.

### 4.4.4 Cálculo de la trayectoria

Como se ha comentado previamente, la línea amarilla presente en el suelo marca la trayectoria a seguir por el robot. El modelo CNN desarrollado en el punto anterior permite identificar la posición de dicha línea en la imagen. En este punto se especifica el proceso empleado en Matlab para traducir la segmentación obtenida por la red en un conjunto de puntos que formen la trayectoria a seguir por el AGV.

La función “semanticseg” de la librería de Deep Learning de Matlab permite segmentar una imagen a través de un modelo CNN. Esta función devuelve como respuesta una matriz categórica unidimensional, de las mismas proporciones que la imagen de entrada, 360x640 en nuestro caso. La Figura 50 muestra un ejemplo de dicha matriz.

C

360x640 categorical

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea |
| 2 | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea |
| 3 | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea |
| 4 | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea |
| 5 | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea |
| 6 | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea |
| 7 | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea |
| 8 | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea |
| 9 | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea |
| 10 | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea |
| 11 | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea |
| 12 | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea |
| 13 | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea |
| 14 | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | Linea | Linea |
| 15 | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | Linea | Linea |
| 16 | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | Linea | Linea |
| 17 | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | NoLinea | Linea | Linea |

*Figura 50: Ejemplo de Matriz categórica obtenida mediante la función semanticseg*

Mediante comparaciones lógicas es posible obtener una máscara similar a las empleadas como etiquetas en el proceso de entrenamiento, como la mostrada en la Figura 48. Con esta se conoce que pixeles de la imagen son ocupados por la línea amarilla.

Para traducir ese conjunto de pixeles a un vector de puntos que indique la trayectoria a recorrer por el AGV se aplica el siguiente criterio:

- Analizar la máscara por vectores fila, que se corresponden con trazos horizontales en la imagen.
- Detectar los puntos de inicio y finalización de los pixeles marcados como línea en dicho vector fila.
- Tomar el punto central entre el inicio y el final como perteneciente a la trayectoria.
- Repetir el procedimiento por cada fila de la imagen hasta obtener un vector que incluya todos los puntos medios.
- Opcionalmente, superponer el conjunto de puntos determinado sobre la imagen original para validar el proceso.

En los Anexos, en el apartado Subfunción 2: Calcular_ptos_medios.m se detalla la función empleada para tal fin. La Figura 51 muestra un ejemplo de la superposición de la trayectoria calculada sobre la imagen original. La trayectoria calculada se marca con puntos verdes.

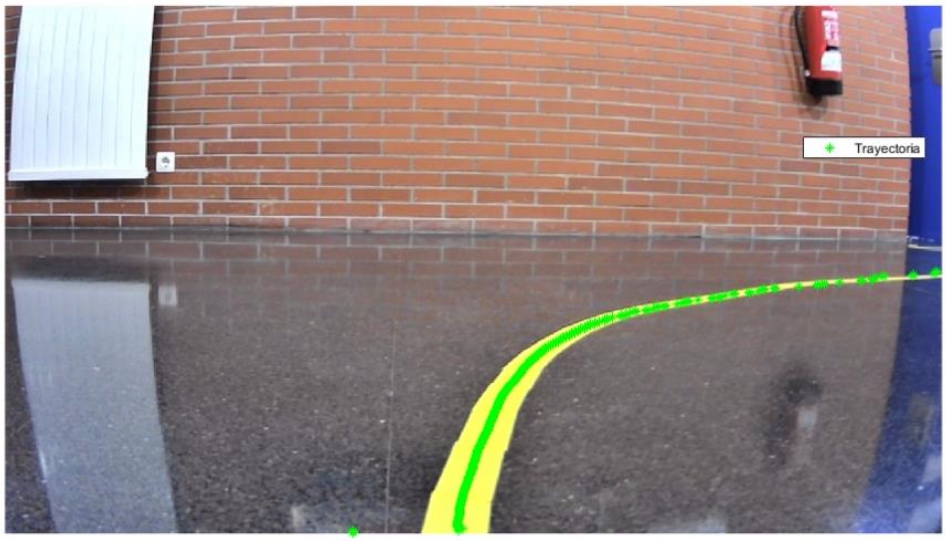



*Figura 51: Trayectoria calculada sobre imagen original*

### 4.4.5 Seguimiento de la trayectoria (Steering)

Una vez obtenida la trayectoria, siguiendo el procedimiento del punto anterior, es necesario implementar el algoritmo de navegación que permita al AGV seguir la trayectoria. Para ello, en el proyecto, se ha tomado como el sistema presentado por Teso et al. en [69].

En primer lugar, se establece un punto de control. Este punto de control se define como el punto de la trayectoria más alejado de la foto. En el sistema implementado, al tratarse de una imagen de 640px de ancho, los puntos medios se encuentran en los pixeles que ocupan la posición 320 en la vertical.

Se define el error de seguimiento de la trayectoria como la diferencia, en pixeles, entre los puntos de la trayectoria y en centro de la imagen. $\bar{T}$ es el vector que contiene la posición de los puntos de la trayectoria. $\bar{E}$ es un vector que contiene el error de cada punto de la trayectoria respecto del centro de la imagen. La ecuación (1) muestra el cálculo del vector $\bar{E}$.

$$\bar{E} = \bar{T} - 320 \tag{1}$$

A partir de los errores de seguimiento se define un punto de control. El punto de control se corresponde con el punto de la trayectoria que presenta mayor error absoluto. Conocido el punto $P$, de la trayectoria, que presenta mayor error se define la compensación a ejercer sobre las ruedas. La Figura 51 muestra el punto de control calculado sobre la trayectoria. La trayectoria es la misma que la presentada en la Figura 50.

El "Steering", es el proceso mediante el que los AGV determinan la cantidad de giro adecuada para el seguimiento de una trayectoria. En el caso del prototipo de AGV del proyecto, no se dispone de control directo sobre la dirección. Por el contrario, es posible manejar cada una de las ruedas de forma independiente, lo que permite al AGV realizar giros.

El sistema de navegación presentado en [69] fija una velocidad base para ambas ruedas. En el algoritmo de navegación desarrollado, se fija dicha velocidad en un 15% de la velocidad máxima de los motores.

El seguimiento de la trayectoria se realiza compensando la velocidad de una de las ruedas, de tal forma que se permita el giro del AGV. La compensación se ejerce sobre la rueda contraria al sentido de giro. Así, en el ejemplo mostrado en la Figura 52, la rueda compensada seria la izquierda.

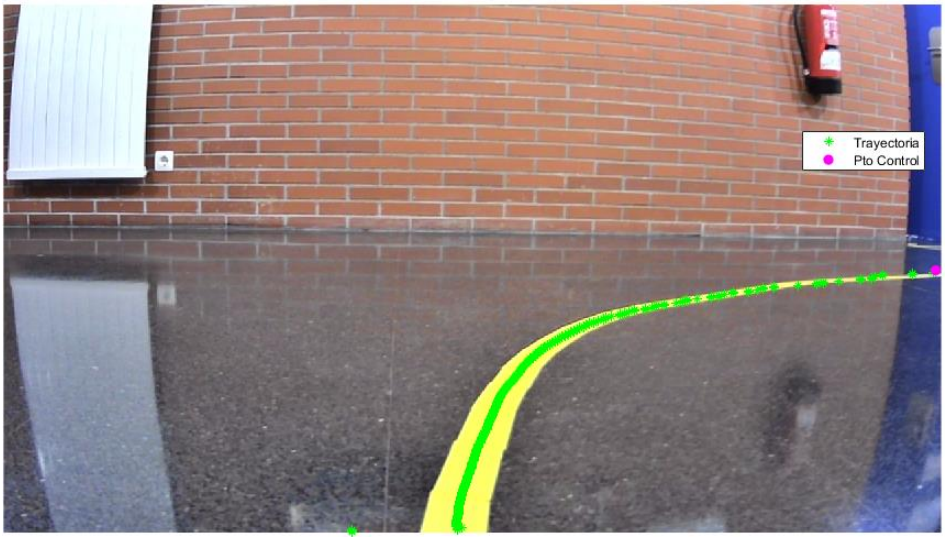



*Figura 52: Punto de control calculado sobre la trayectoria*

La compensación consiste en aumentar la velocidad de la rueda en función del error del punto de control. El error máximo sucede cuando el punto de control se encuentra en el extremo de la imagen. Dicho error tendría un valor de 320px. El factor de compensación se define como el tanto por uno de error sobre el error máximo, como muestra la ecuación (2).

$$Steer = \frac{Error_{ptoControl}}{Error_{Max}} \quad (2)$$

Finalmente, la velocidad de la rueda compensada se obtiene al incrementar la velocidad nominal linealmente con el "Steer" calculado.

$$Vel_{RueadaCompesnada} = V_{nom} \cdot (1 + Steer) \quad (3)$$

El código desarrollado para tal fin, y descrito en el Anexo: Subfunción 3: Calcular_Vel, distingue cual es la rueda a compensar aplicando la función signo al error del punto de control.

El último paso es ajustar el valor de las velocidades calculadas a la representación del Arduino. Como se comenta en la sección destinada al hardware del Arduino (4.4.7 ), los motores se controlan mediante los puertos PWM de la placa. Arduino ajusta los valores del PWM entre el 0 y el 255. En el sistema de control planteado las velocidades se definen en tanto por uno de la velocidad máxima del motor. Por tanto, antes de enviar la trama, es necesario escalar las velocidades calculadas al rango 0-255.

### 4.4.6 Comunicación Serie

Con la velocidad de cada una de las ruedas determinada, el último paso que debe de realizar el algoritmo de navegación es transmitir las consignas de velocidad al microcontrolador. Como se ha descrito en los objetivos la comunicación entre el PC de procesamiento de imagen y el microcontrolador del AGV se realiza mediante puerto serie. Por tanto, es necesario definir la estructura de la trama.

La trama enviada por el puerto serie es una cadena de caracteres que contiene en primer lugar un identificador. En la Tabla 1 se muestra el resumen de los identificadores permitidos en la trama y su acción correspondiente.

*Tabla 1: Identificadores de la Trama Serie*

| IDENTIFICADOR | ACCIÓN |
|---|---|
| 'V' | Indica que los caracteres siguientes se corresponden con la consigna de velocidad de las ruedas. |
| 'S' | Indica al microcontrolador que debe parar el AGV. |
| 'G' | Indica al Arduino que debe establecer la velocidad nominal en ambas ruedas. |
| 'T' | Sin acción asignada. Se emplea en tareas de depuración. |

Cuando se envía el identificador 'V', se completa la trama enviando la velocidad de la rueda izquierda en primer lugar y después la de la derecha. Ambas velocidades vienen precedidas por un prefijo para poder separarlas adecuadamente. Un ejemplo de trama, que actualiza la velocidad de la rueda izquierda a 75 y la de la derecha a 60 es:

'VL75R60'

Los programas incluidos en los Anexos: Subfunción 4: Send_dutty_V2.m y Código de control de Motores en Arduino muestran cómo construir y extraer información de dicha trama.

### 4.4.7 Diseño del nuevo Hardware del AGV

Como se ha descrito en el apartado 4.2.1 Prototipo de AGV , el hardware original del robot Pioneer 3-DX es sustituido por otro de diseño propio. En este apartado se define el hardware desarrollado para el control de los motores del AGV.

Originalmente el robot cuenta con un par de Motores de corriente continua (DC) que operan a una tensión de 12V. Por tanto, el primer paso del diseño es incorporar unas baterías capaces de administrar dicha tensión. En la Figura 53 se muestra un ejemplo de las baterías empleadas.


*Figura 53: Baterías de 12V que alimentan al AGV*

Un microcontrolador de Arduino UNO, solo es capaz de dar tensiones en el rango 0-5V gracias a sus puertos PWM. Este rango de tensión no es suficiente para poder manejar los motores de forma apropiada por lo que es necesario emplear un Driver. Para tal fin se opta por el driver L298N, utilizado comúnmente con las placas de Arduino para este tipo de aplicaciones.

*Figura 54: Driver L298N*

Este Driver tiene 2 funciones. En primer lugar, permitir controlar los motores en el rango de operación de los mismo 0-12V. Por otro lado, también funciona como un transformador que permite obtener una salida de 5V de DC. Esta salida se emplea para alimentar el Arduino, ahorrando la necesidad de emplear un segundo componente para alimentar al microcontrolador a través de la batería.

El driver L298N contiene dos puentes en H en su placa, por lo que es capaz de manejar 2 motores simultáneamente. Cada puente en H tiene tres pines de control a conectar al Arduino. Los dos primeros, permiten controlar los interruptores del puente 2 a 2 mientras que el tercer pin, denominado “Enable”, es el pin al que se conecta la salida PWM del Arduino y marca la tensión

de salida a aplicar al motor. En el Anexo: Esquema del rediseño de hardware basado en la placa Arduino Uno se detalla el esquema eléctrico de las conexiones.

### 4.4.8 Programación del Arduino

Las placas de Arduino son microcontroladores que están orientados a trabajar como sistemas embebidos. Por tanto, su programación consiste en una serie de instrucciones que se ejecutan cíclicamente mientras la placa se mantenga alimentada.

El algoritmo de control desarrollado se puede dividir en tres apartados:

- Bloque de inicialización
- Bucle principal
- Rutina de servicio de interrupción (RSI)

En el bloque de inicialización se establece la comunicación por el puerto serie, asignando la velocidad (baudrate) y el resto de las propiedades. Es importante destacar que, en el algoritmo de navegación implementado en Matlab, a la hora de configurar el puerto serie, es imprescindible configurar la misma velocidad.

Al finalizar un ciclo completo del bucle principal, el Arduino comprueba si ha saltado alguna interrupción. Si la interrupción tiene activa su “flag” el Arduino pasa a ejecutar la RSI. Una RSI es un fragmento de código, que se ejecuta exclusivamente cuando se dan las circunstancias adecuadas. En este caso, se emplea la RSI del puerto serie implementada en el lenguaje del Arduino.

Mediante el comando “serialEvent” es posible generar una función que se ejecute como RSI asociada a la interrupción del puerto serie. En este fragmento de código se lee la información almacenada en el “buffer” del puerto serie y se almacena en una variable, de tal forma que la trama leída pueda ser procesada en el bucle principal.

En el bucle principal se comprueba si ha llegado una nueva trama. En el momento en el que esto sucede se identifica la etiqueta de la trama, leída por la RSI y se opera en consecuencia. El detalle del funcionamiento del programa está en el Anexo: Código de control de Motores en Arduino

# 5 Pruebas y Resultados

En esta sección se detalla el diseño de las pruebas llevadas a cabo para evaluar los sistemas que componen el algoritmo de localización y navegación desarrollado. Además, se exponen los resultados obtenidos en dichas pruebas.

## 5.1 Diseño de las pruebas

### 5.1.1 Prueba de precisión de las CNN

Para evaluar la precisión del segmentado de las redes neuronales se genera una pequeña muestra de 64 imágenes. Las imágenes son capturadas por la cámara instalada en el AGV de tal forma que la altura esté ajustada. En las imágenes se muestran diferentes trayectorias ante las que puede encontrarse el AGV. La Figura 55 Muestra 3 de las 64 imágenes tomadas para realizar las pruebas de precisión.

*Figura 55: Muestra de las 64 imágenes tomadas para las pruebas de precisión*

Esta prueba consistirá en tomar el conjunto de 64 imágenes, obtener la máscara de segmentación correspondiente y compararla con el etiquetado original. Para la comparación se emplearán dos métodos: El primero es visual, emplea la función “imshowpair” que permite destacar visualmente las discrepancias entre dos imágenes. El segundo es un método de comparación numérico. Utiliza el índice de Jaccard como métrica de la precisión de los modelos de CNN. Este índice mide la intersección sobre la unión de la segmentación frente a la etiqueta para cada clase y calcula la media [70]. Así, el índice de Jaccard tiene en cuenta tanto los falsos positivos como los falsos negativos para cada clase.

### 5.1.2 Prueba de Velocidad de Ejecución de las CNN

Mediante esta prueba se busca conocer el tiempo de ejecución total de algoritmo de navegación y el peso de la ejecución del modelo CNN en dicho coste computacional.

Para ello se emplean las funciones “tic” y “toc” de Matlab que permiten medir los tiempos de ejecución. Además del tiempo de ejecución total de un ciclo del algoritmo se mide por separado los tiempos empleados en realizar las siguientes tareas:

- Segmentar la Imagen
- Calcular la velocidad
- Transmitir la trama por puesto serie
- Tomar la imagen a través de la webcam
- Determinar la trayectoria
- Calcular el punto de control

El objetivo es determinar cuál es el cuello de botella de algoritmo y demostrar si es viable emplearlo en el manejo del AGV. Se seguirá la afirmación propuesta Paszke et al. [67] que estiman una tasa de 10 fps como valor mínimo para tareas de localización mediante visión.

### 5.1.3 Prueba de Seguimiento de la trayectoria

La primera parte de la prueba se basa en un análisis cualitativo que consiste en probar la capacidad del AGV de seguir la trayectoria marcada por la línea amarilla. Su valoración consistirá en valorar si el AGV ha sido capaz de alcanzar el final de la trayectoria y de la calidad del recorrido descrito por el robot.

En segundo lugar, se observará la gráfica de puntos de control acumulados. Esta gráfica representa la posición que ocupa el punto de control en cada ciclo de ejecución. El objetivo es observar la capacidad del robot para centrarse en el seguimiento de la trayectoria.

Para realizar la prueba se emplearán los 2 circuitos de pruebas descritos en 4.2.9 Circuitos de ensayo. En la primera prueba se empleará el circuito curvo, mientras que en el segundo el circuito recto.

## 5.2 Análisis de Resultados

### 5.2.1 Análisis de la precisión de las CNN

El primer modelo evaluado, es el obtenido a partir de la arquitectura DeepLabV3. Este modelo ha sido reentrenado empleando el conjunto de imagenes del pasillo de la escuela de ingeniería. Para realizar este análisis se parte de la imagen de la Figura 56.

*Figura 56: Imagen base para el análisis de la segmentación*

La Figura 57 muestra la comparación visual aplicando el comando “imshowpair”. En negro se muestran los pixeles clasificados como línea amarilla tanto en la etiqueta como en el segmentado del modelo. Los colores verde y magenta representan los pixeles mal clasificados por el modelo. Puede observarse que, en líneas generales, la clasificación realizada por el modelo es aceptable.

Adicionalmente se calcula el índice de Jaccard sobre el segmentado obtenido por el modelo CNN. Para la clase “No línea” se obtiene un índice de 0.99 lo que implica un índice de acierto casi exacto. En cambio, para la clase “Línea” el valor es de 0.7. Es un valor lo suficientemente alto como para validad el modelo. Además, en su mayoría, los pixeles mal clasificados se encuentran en la frontera entre la trayectoria y el entorno.

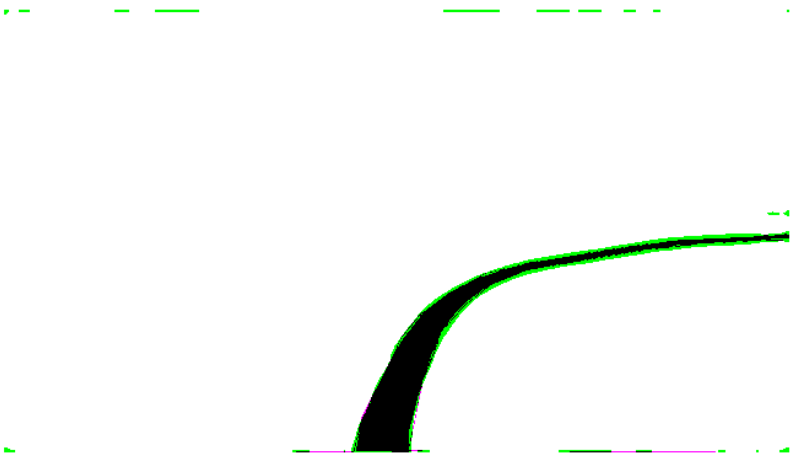


*Figura 57: Resultado de la comparación del modelo DeepLAbV3 vs la etiqueta original*

Tomando la misma imagen base, se comprueba la precisión del modelo desarrollado basado en la arquitectura ED-CNN. La Figura 58 muestra el resultado obtenido por la comparación mediante “imshowpair”. Se puede observar que el error alrededor de la trayectoria es sensiblemente mayor. Además, existe zona de falso positivo debido al brillo del radiador de la parte superior izquierda de la imagen.

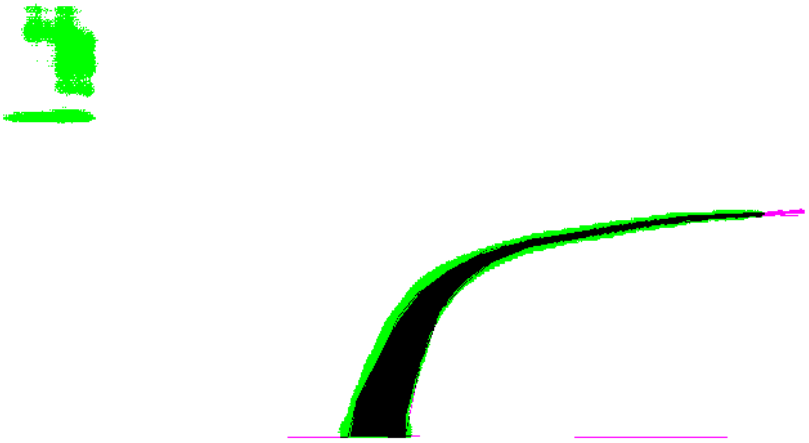


*Figura 58: Resultado de la comparación del modelo ED-CNN vs etiqueta original*

Aplicando el índice Jaccard se obtiene una precisión de 0.98 para la clase “No Línea” y del 0.5 para “Línea”. El valor de este índice indica que la calidad del segmentado no es lo suficientemente alta como para aplicarse en la navegación autónoma.

Sin embargo, en el cálculo de la trayectoria, el tercio superior de la imagen es descartado. Realizando un análisis de los falsos positivos emitidos por el modelo a lo largo de la base de datos se observa que la mayoría de ellos se deben a objetos de las paredes con un alto brillo. Por tanto, a efectos de cálculo de la trayectoria el modelo puede considerarse como valido.

Finalmente, a modo de resumen, la Tabla 2 muestra los valores del índice de Jaccard obtenidos para cada modelo.

*Tabla 2: Índices de Jaccard obtenidos por los modelos DeepLabV3 y ED-CNN*

| **ÍNDICE JACCARD** | DEEPLABV3 | ED-CNN |
|---|---|---|
| NO LÍNEA | 0.99 | 0.98 |
| LÍNEA | 0.7 | 0.5 |

### 5.2.2 Análisis de la velocidad de ejecución de las CNN

Se realiza una simulación haciendo recorrer al AGV uno de los circuitos. En una variable interna, denominada Telemetría, se almacenan los datos de tiempos de ejecución medidos durante cada ciclo, para su posterior procesado. En la Figura 59 se muestra una gráfica en la que se puede apreciar el tiempo invertido en ejecutar cada una de las tareas principales del algoritmo. El procesado de la imagen mediante la segmentación es notablemente más lento que el resto de los procesos.

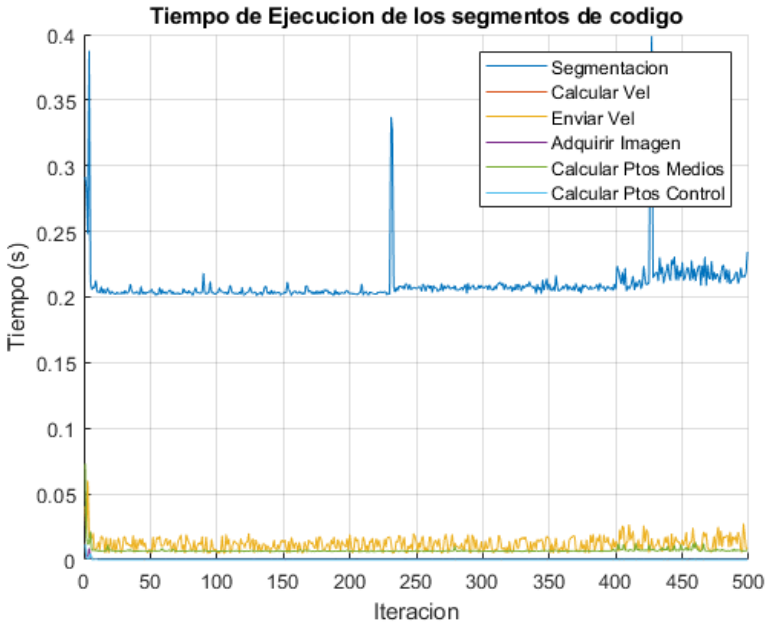



*Figura 59: Tiempos de ejecución de los procesos con el modelo DeepLabV3*

Para poner en perspectiva dichos tiempos se presenta una nueva gráfica, en la Figura 60, que muestra que porcentaje de tiempo de ejecución de un ciclo ha sido invertido en cada tarea. En dicha gráfica se puede observar cómo aproximadamente el 90% del tiempo de ejecución se invierte en el procesado de la imagen.

El tiempo medio de ejecución de un ciclo es de 0.2359 segundos mientras que la mediana se sitúa en 0.2320 La desviación típica se sitúa en 0.0220.

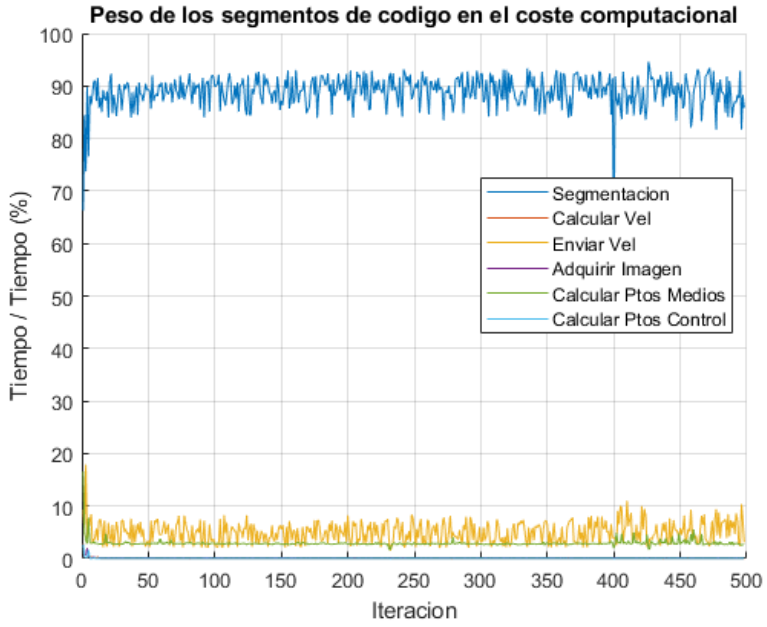



*Figura 60: Porcentaje del tiempo de ejecución total empelado en cada tarea con el modelo DeepLabV3*

Realizando el mismo análisis para el modelo basado en la arquitectura ED-CNN se obtienen los tiempos de ejecución mostrados en la Figura 61. De forma similar al modelo anterior el segmentado de la imagen es más lento que el resto de los procesos. Sin embargo, en conjunto, esta arquitectura es más rápida que la anterior.

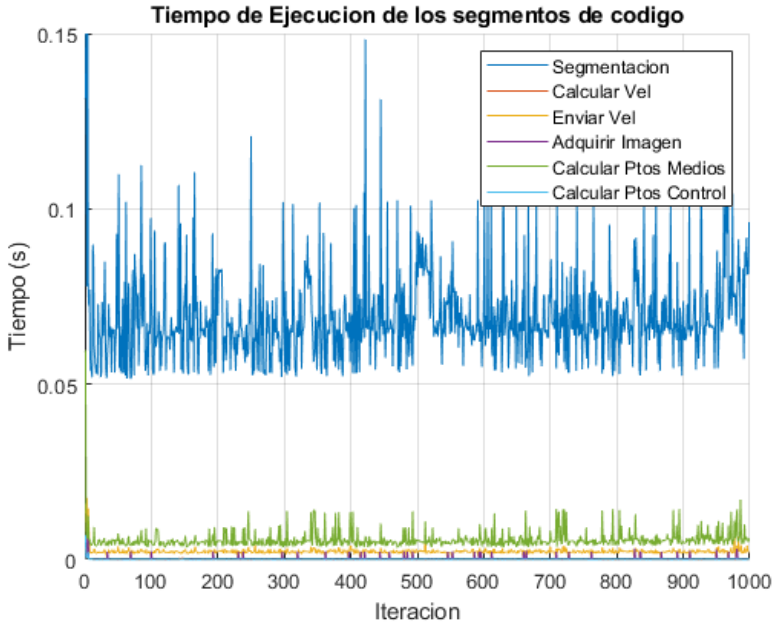



*Figura 61: Tiempos de ejecución de los procesos con el modelo ED-CNN*

Al igual que con el modelo anterior se muestra la gráfica en la que se representa el peso de cada una de las tareas en el tiempo total de ejecución. Nuevamente el procesado de la imagen ocupa el 90% de un único ciclo.

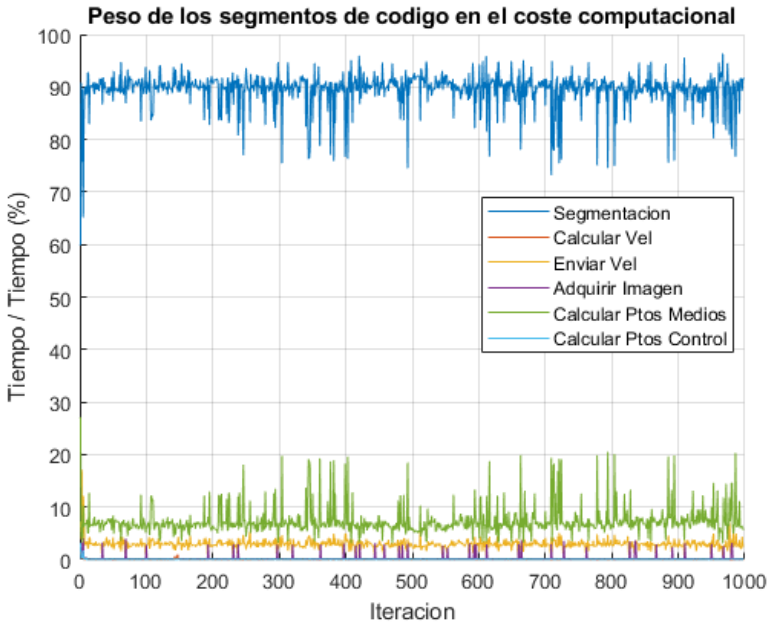



*Figura 62: Porcentaje del tiempo de ejecución total empelado en cada tarea con el modelo ED-CNN*

El tiempo medio de ejecución de un ciclo es de 0.0773 segundos mientras que la mediana se sitúa en 0.0744. La desviación típica se sitúa en 0.0139.

La Tabla 3 muestra un resumen de lo tiempos promedios de ejecución de un ciclo del algoritmo de navegación empleando cada uno de los modelos de CNN.

*Tabla 3: Tiempos de ejecución*

| | **Tiempo Medio** | **Mediana** | **Desviación típica** |
|---|---|---|---|
| DeepLabV3 | 0.2359 | 0.2320 | 0.0220 |
| ED-CNN | 0.0773 | 0.0744 | 0.0139 |

El tiempo de ejecución promedio del modelo basado en ED-CNN es un 67% más rápido que el basado en DeepLabV3. Además, con este tiempo se consigue una tasa de 12 ciclos por segundo. Como en cada ciclo se procesa un “frame”, se obtiene una tasa de 12 fps empleando el modelo ED-CNN. Esta tasa es superior al mínimo especificado por propuesta Paszke et al. [67] por lo que el dicho modelo queda validado.

### 5.2.3 Análisis del seguimiento de la trayectoria

Tras realizar un ensayo en el que el AGV trata de seguir la trayectoria en el circuito recto, se obtiene la gráfica de la Figura 63. Dicha gráfica contiene la representación de la posición en la que se encontraba el punto de control en cada una de las iteraciones. Al tratarse de un recorrido recto, los puntos deberían estar agrupados cerca del punto medio de la imagen (320px).

Sin embargo, puede observarse como los puntos de control aparecen muy dispersos y en su mayoría se encuentran en los bordes de la zona de cálculo. Esto es debido a que la velocidad de ejecución del algoritmo no es lo suficientemente rápida. Las nuevas consignas de velocidad tardan en llegar al microcontrolador por lo que el AGV mantiene unas velocidades incorrectas en los motores. Visualmente es apreciable como el robot "culea" sobre sí mismo realizando pequeños trazos en 'S' sobre la trayectoria en lugar de recorrerla por encima.

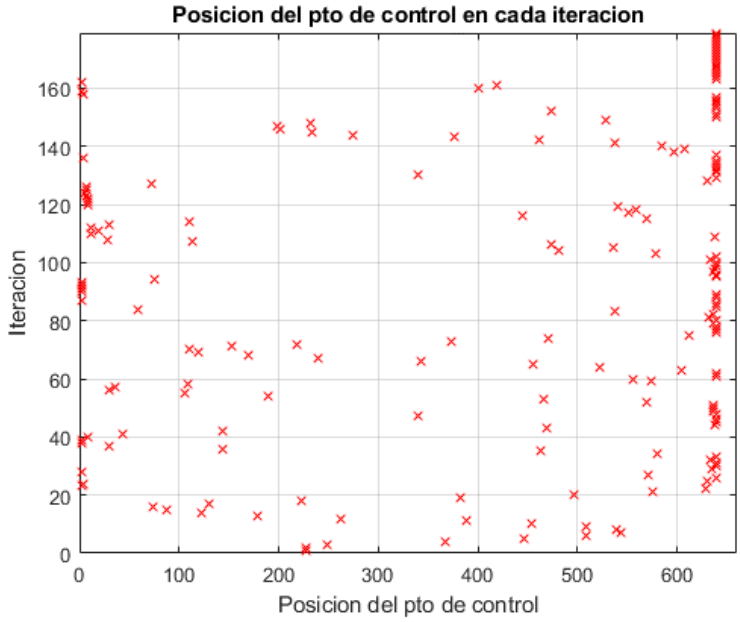



*Figura 63: Mapa de puntos de control con el modelo DeepLabV3*

A modo de comparación, la Figura 64 muestra el mapa obtenido al realizar el ensayo empleando el modelo ED-CNN. Puede observarse que este modelo es lo suficientemente rápido como para mantener al AGV sobre la trayectoria. EL conjunto de puntos se encuentra entre los valores de 300 y 400 px lo que indica que el Robot se mantiene una trayectoria relativamente recta.

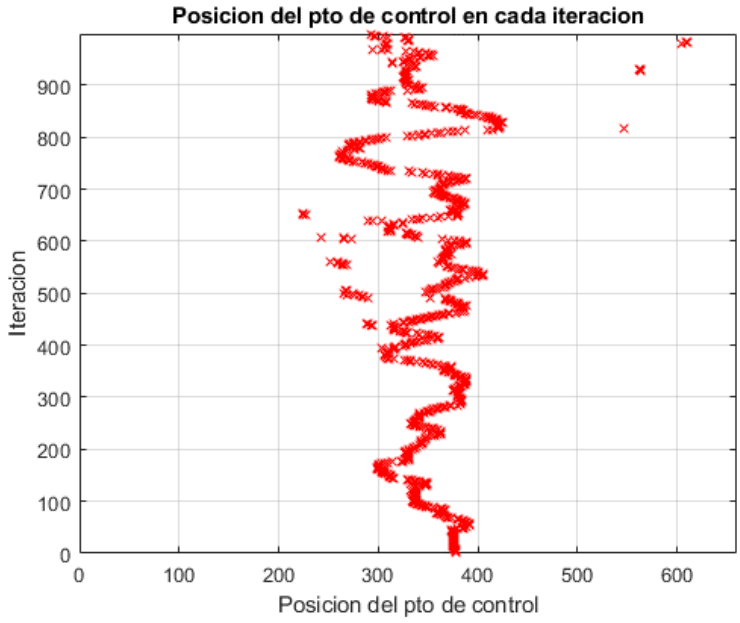



*Figura 64: Mapa de puntos de control con el modelo ED-CNN*

En virtud de los resultados obtenidos el modelo seleccionado para formar parte del algoritmo de navegación y localización es el basado en la arquitectura ED-CNN. En los enlaces adjuntos se muestran unos videos en los que se puede apreciar el comportamiento del AGV al tratar de seguir la trayectoria en cada uno de los modelos propuestos. La diferencia del comportamiento entre ambos casos corrobora la decisión tomada.

*AGV recorriendo circuito curvo con algoritmo de navegación y localización basado en DeeplabV3*

*AGV recorriendo circuito curvo con algoritmo de navegación y localización basado en ED-CNN*

# 6 Conclusiones y Trabajos futuros

El apartado dedicado a las conclusiones aborda la consecución de los objetivos marcados al inicio del proyecto, así como los conocimientos adquiridos durante la realización de éste. Por otro lado, el apartado dedicado a trabajos futuros expone algunas ideas que se quedaron fuera del proyecto por falta de encaje o tiempo y otras líneas de trabajo futuras que pueden resultar de interés.

## 6.1 Conclusiones

El proyecto ha mostrado los pasos necesarios para desarrollar un algoritmo de navegación y localización. Este proyecto ha contribuido al análisis de los diferentes métodos de navegación y localización existentes en la actualidad y a acotar las áreas en las que el prototipo del departamento puede operar.

A raíz del análisis del estado del arte, se ha observado que la tendencia actual lleva a los AGV industriales a emplear estrategias de navegación y localización que no tengan ninguna dependencia de terceras infraestructuras. Así, a pesar de que el sistema planteado en el desarrollo del proyecto es ampliamente flexible, pues apenas se tardan unos minutos en rehacer la trayectoria, sería necesario tratar de implementar un sistema de navegación totalmente independiente.

Las dos principales aportaciones del proyecto han sido: Por un lado, la elaboración del hardware de control para el AGV y la programación del microcontrolador y, por otro lado, el desarrollo de la CNN. La aportación de estos dos logros destaca sobre los objetivos alcanzados en el desarrollo del proyecto. La nueva infraestructura del AGV independiza el control de los motores del algoritmo de navegación y localización. El robot es compatible con cualquier nuevo algoritmo que mantenga la estructura de las consignas y la comunicación serie.

Por otro lado, la CNN desarrollada puede ser ejecutada en otros sistemas o AGVs. En el propio laboratorio del departamento existen otros modelos empleados por alumnos de grado que han hecho uso de dicho modelo en sus respectivos AGV con buenos resultados. Además, la metodología de entrenamiento de los modelos desarrollada es adaptable a nuevos enfoques. Lo cual permite el entrenamiento de nuevos modelos de CNN que apliquen un principio de detección diferente. Solo sería necesario definir la nueva arquitectura y definir un conjunto de datos de imágenes acorde al problema que se pretende solucionar.

## 6.2 Trabajo Futuro y Nuevas líneas de investigación

Dentro de las posibilidades que ofrecía el presente proyecto algunas de las ideas descartadas, presentan pequeñas mejoras que permitirían desarrollar un sistema de localización y navegación más preciso y robusto.

En primer lugar, el seguimiento de la trayectoria se basa en un sistema que emplea un único punto de la trayectoria. El procesamiento de la imagen es el cuello de botella de algoritmo. Emplear únicamente un punto en lugar de toda la trayectoria es un infrautilizamiento del sistema de visión desarrollado. Por tanto, implementar un algoritmo de control que haga uso de más puntos aumentaría la eficiencia del algoritmo.

Otro aspecto que no se ha tenido en cuenta es la detección activa de obstáculos. Con el objetivo de mejorar el prototipo de AGV a una versión más cercana a un mecanismo real es necesario implementar un sistema de detección de obstáculos. Para ello existen diferentes alternativas. Los AGV vienen incorporados con una serie de sensores de ultrasonidos en su frontal. Por otro lado, el laboratorio cuenta con sensores tipo LiDAR. Ambos pueden emplearse en tareas de localización de obstáculos. Otra posibilidad es mejorar el sistema de visión de tal forma que sea capaz de detectar y evitar obstáculos.

Finalmente, una solución que permitiría aumentar considerablemente la velocidad de procesamiento del sistema de visión, a la par que permitir al AGV trabajar con requisitos de tiempo real, sería el empleo de un sistema embebido de procesamiento de imagen. El laboratorio cuenta con una tarjeta NVIDIA Jetson, como la de la Figura 65.


*Figura 65: NVIDIA Jetson*

Este tipo de dispositivos permiten la ejecución de modelos de CNN a unas velocidades mucho mayores. Además, al tratarse de sistemas embebidos, cuentan con una versión simplificada de Ubuntu como sistema operativo lo que le permite controlar periféricos como la cámara web o comunicarse con el microcontrolador. Esto lo convierte en un candidato ideal para sustituir al PC como sistema de procesamiento de imagen.

# 7 Bibliografía

[1] M. De Ryck, M. Versteyhe and F. Debrouwere, "'Automated guided vehicle systems, state-of-the-art control algorithms and techniques," *J.Manuf.Syst.*, vol. 54, pp. 152-173.

[2] G.V. Research, "'Automated guided vehicles market,", pp. 255-270.

[3] D. Bechtsis, N. Tsolakis, D. Vlachos and E. Iakovou, "'Sustainable supply chain management in the digitalisation era: The impact of Automated Guided Vehicles," *J.Clean.Prod.*, vol. 142, pp. 3970-3984.

[4] R.Y. Zhong, X. Xu, E. Klotz and S.T. Newman, "'Intelligent Manufacturing in the Context of Industry 4.0: A Review," *Engineering*, vol. 3, no. 5, pp. 616-630.

[5] I. Draganjac, T. Petrović, D. Miklić, Z. Kovačić and J. Oršulić, "'Highly-scalable traffic management of autonomous industrial transportation systems," *Robot.Comput.Integrated Manuf.*, vol. 63, pp. 101915.

[6] M. De Ryck, M. Versteyhe and K. Shariatmadar, "'Resource management in decentralized industrial Automated Guided Vehicle systems," *J.Manuf.Syst.*, vol. 54, JAN, pp. 204-214.

[7] X. Jia and M.Q.-. Meng, "'A survey and analysis of task allocation algorithms in multi-robot systems," *2013 IEEE International Conference on Robotics and Biomimetics, ROBIO 2013*, pp. 2280-2285.

[8] F. Gul, W. Rahiman and S.S. Nazli Alhady, "'A comprehensive study for robot navigation techniques," *null*, vol. 6, no. 1, pp. 1632046.

[9] S. G. Anavatti, S. L. Francis and M. Garratt, "'Path-planning modules for Autonomous Vehicles: Current status and challenges," *2015 International Conference on Advanced Mechatronics, Intelligent Manufacture, and Industrial Automation (ICAMIMIA)*, pp. 205-214.

[10] V. Kunchev, L. Jain, V. Ivancevic and A. Finn, "'Path Planning and Obstacle Avoidance for Autonomous Mobile Robots: A Review,", Systems, pp. 537-544.

[11] Q.S. Kabir and Y. Suzuki, "'Increasing manufacturing flexibility through battery management of automated guided vehicles," *Comput.Ind.Eng.*, vol. 117, pp. 225-236.

[12] A. Vale, R. Ventura, P. Lopes and I. Ribeiro, "'Assessment of navigation technologies for automated guided vehicle in nuclear fusion facilities," *Robot.Auton.Syst.*, vol. 97, NOV, pp. 153-170.

[13] P. Yin, W. Li and Y. Duan, "'Combinatorial inertial guidance system for an automated guided vehicle," *2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC)*, pp. 1-6.

[14] J. Song, "'Electromagnetic Induction Sensor of Navigation System for Spraying Robot,", 1, Melbourne, Australia, pp. 175-181.

[15] S. E. M. Bajestani and A. Vosoughinia, "'Technical report of building a line follower robot," *2010 International Conference on Electronics and Information Engineering*, vol. 1, pp. V1-5.

[16] J. Choi and B. Choi, "'Design of self-localization based autonomous driving platform for an electric wheelchair," *2017 11th Asian Control Conference (ASCC)*, pp. 465-466.

[17] S. Lee and H. Yang, "'Navigation of automated guided vehicles using magnet spot guidance method," *Robot.Comput.Integrated Manuf.*, vol. 28, no. 3, pp. 425-436.

[18] A.S. Savelius, "'Indoor positioning technologies for industrial heavy-duty vehicle application," *Indoor positioning technologies for industrial heavy-duty vehicle application*.

[19] R. Lin, H. Huang and M. Li, "'An automated guided logistics robot for pallet transportation," *Assem.Autom.*, vol. 41, no. 1, FEB 19, pp. 45-54.

[20] J.R. Souza, G. Pessin, G.B. Eboli, C.C. Mendes, F.S. Osório and D.F. Wolf, "'Vision and gps-based autonomous vehicle navigation using templates and artificial neural networks,", pp. 280-285.

[21] Z. Rozsa and T. Szirany, "'Obstacle Prediction for Automated Guided Vehicles Based on Point Clouds Measured by a Tilted LIDAR Sensor," *IEEE Trans.Intell.Transp.Syst.*, vol. 19, no. 8, AUG, pp. 2708-2720.

[22] D. Teso-Fz-Betono, E. Zulueta, U. Fernandez-Gamiz, I. Aramendia and I. Uriarte, "'A Free Navigation of an AGV to a Non-Static Target with Obstacle Avoidance," *Electronics*, vol. 8, no. 2, FEB, pp. 159.

[23] R. G. Yudanto and F. Petré, "'Sensor fusion for indoor navigation and tracking of automated guided vehicles," *2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pp. 1-8.

[24] S.C. Calvert, W.J. Schakel and van Lint, J. W. C, "'Will Automated Vehicles Negatively Impact Traffic Flow?" *Journal of advanced transportation*, vol. 2017, Sep 28, pp. 1-17.

[25] A. Gupta, A. Anpalagan, L. Guan and A.S. Khwaja, "'Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues," *Array*, vol. 10, pp. 100057.

[26] J. Fayyad, M.A. Jaradat, D. Gruyer and H. Najjaran, "'Deep Learning Sensor Fusion for Autonomous Vehicle Perception and Localization: A Review," *Sensors*, vol. 20, no. 15, AUG, pp. 4220.

[27] M. Al-Qizwini, I. Barjasteh, H. Al-Qassab and H. Radha, "'Deep learning algorithm for autonomous driving using GoogLeNet," *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 89-96.

[28] C. Wen, X. Sun, J. Li, C. Wang, Y. Guo and A. Habib, "'A deep learning framework for road marking extraction, classification and completion from mobile laser scanning point clouds," *ISPRS-J.Photogramm.Remote Sens.*, vol. 147, JAN, pp. 178-192.

[29] J. Zhang, L. Peng, W. Feng, Z. Ju and H. Liu, "'Human-AGV Interaction: Real-Time Gesture Detection Using Deep Learning,", Applications, pp. 231-242.

[30] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, Li Fei-Fei and S. Savarese, "'Social LSTM: Human Trajectory Prediction in Crowded Spaces," *2016 Ieee Conference on Computer Vision and Pattern Recognition (Cvpr)*, pp. 961-971.

[31] G.E. Hinton, "'Deep belief networks," *Scholarpedia*, vol. 4, no. 5, pp. 5947.

[32] C. Li, Y. Wang, X. Zhang, H. Gao, Y. Yang and J. Wang, "'Deep Belief Network for Spectral–Spatial Classification of Hyperspectral Remote Sensor Data," *Sensors*, vol. 19, pp. 204.

[33] P. Baldi, "'Autoencoders, Unsupervised Learning, and Deep Architectures," *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, vol. 27, pp. 37-49.

[34] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio and P. Manzagol, "'Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion," *J.Mach.Learn.Res.*, vol. 11, DEC, pp. 3371-3408.

[35] J. Briot, G. Hadjeres and F. Pachet, "'Deep Learning Techniques for Music Generation - A Survey,".

[36] A. Krizhevsky, I. Sutskever and G.E. Hinton, "'Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097-1105.

[37] R. Mash, N. Becherer, B. Woolley and J. Pecarina, "'Toward aircraft recognition with convolutional neural networks,", 2016.

[38] F. Sultana, A. Sufian and P. Dutta, "'Advancements in Image Classification using Convolutional Neural Network," *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, pp. 122-129.

[39] Y. Li, F. Cui, X. Xue and J. Cheung-Wai Chan, "'Coarse-to-fine salient object detection based on deep convolutional neural networks," *Signal Process Image Commun*, vol. 64, pp. 21-32.

[40] A. Singh, R. Sarkhel, N. Das, M. Kundu and M. Nasipuri, "'A Skip-Connected Multi-column Network for Isolated Handwritten Bangla Character and Digit Recognition," *Sensing and Imaging*, vol. 21.

[41] P. Kim, "'Matlab deep learning," *With machine learning, neural networks and artificial intelligence*, vol. 130, pp. 21.

[42] M. Yani, S. Irawan and M.T. S.T., "'Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail," *Journal of Physics: Conference Series*, vol. 1201, pp. 012052.

[43] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis, J. Ngadiuba, M. Pierini, N. Tran and Z. Wu, "'Fast inference of deep neural networks in FPGAs for particle physics," *Journal of Instrumentation*, vol. 13.

[44] H.A. Kholerdi, N. TaheriNejad and A. Jantsch, "'Enhancement of Classification of Small Data Sets Using Self-awareness - An Iris Flower Case-Study," *2018 Ieee International Symposium on Circuits and Systems (Iscas)*.

[45] O. M. Parkhi, A. Vedaldi, A. Zisserman and C. V. Jawahar, "'Cats and dogs," *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3498-3505.

[46] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "'Going deeper with convolutions," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1-9.

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

[48] W. Nash, T. Drummond and N. Birbilis, "'A review of deep learning in the study of materials degradation," *npj Materials Degradation*, vol. 2.

[49] Z. Zhao, P. Zheng, S. Xu and X. Wu, "'Object Detection With Deep Learning: A Review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212-3232.

[50] J. Dai, Y. Li, K. He and J. Sun, "'R-FCN: Object Detection via Region-based Fully Convolutional Networks," *Advances in Neural Information Processing Systems 29 (Nips 2016)*, vol. 29.

[51] J. Sang, Z. Wu, P. Guo, H. Hu, H. Xiang, Q. Zhang and B. Cai, "'An Improved YOLOv2 for Vehicle Detection," *Sensors*, vol. 18, no. 12, DEC, pp. 4272.

[52] M. Barreiros, D. Dantas, L. Silva, S. Ribeiro and A. Barros, "'Zebrafish tracking using YOLOv2 and Kalman filter," *Scientific Reports*, vol. 11, pp. 3219.

[53] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou and G. Cottrell, "'Understanding Convolution for Semantic Segmentation," *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1451-1460.

[54] A. A. Assidiq, O. O. Khalifa, M. R. Islam and S. Khan, "'Real time lane detection for autonomous vehicles," *2008 International Conference on Computer and Communication Engineering*, pp. 82-88.

[55] R. Javanmard, A. H. Zabbah, M. Karimi and K. Jeddisaravi, "'Line Following Autonomous Driving Robot using Deep Learning," *2020 6th Iranian Conference on Signal Processing and Intelligent Systems (ICSPIS)*, pp. 1-5.

[56] G. Yao, T. Lei and J. Zhong, "'A review of Convolutional-Neural-Network-based action recognition," *Pattern Recog.Lett.*, vol. 118, pp. 14-22.

[57] A. Gurghian, T. Koduri, S. V. Bailur, K. J. Carey and V. N. Murali, "'DeepLanes: End-To-End Lane Position Estimation Using Deep Neural Networks," *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 38-45.

[58] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu and R. Cheng-Yue, "'An empirical evaluation of deep learning on highway driving," *arXiv preprint arXiv:1504.01716*.

[59] Y. Huang, S. Chen, Y. Chen, Z. Jian and N. Zheng, "'Spatial-Temproal Based Lane Detection Using Deep Learning,", Innovations, pp. 143-154.

[60] D. Neven, B. D. Brabandere, S. Georgoulis, M. Proesmans and L. V. Gool, "'Towards End-to-End Lane Detection: an Instance Segmentation Approach," *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 286-291.

[61] N. Garnett, R. Cohen, T. Pe'er, R. Lahav and D. Levi, "'3D-LaneNet: end-to-end 3D multiple lane detection,", pp. 2921-2930.

[62] X. Pan, J. Shi, P. Luo, X. Wang and X. Tang, "'Spatial as deep: Spatial cnn for traffic scene understanding,", vol. 32, no. 1.

[63] K. Behrendt and R. Soussan, "'Unsupervised labeled lane markers using maps,", pp. 0.

[64] L. Chen, Y. Zhu, G. Papandreou, F. Schroff and H. Adam, "'Encoder-decoder with atrous separable convolution for semantic image segmentation,", pp. 801-818.

[65] G.J. Brostow, J. Fauqueur and R. Cipolla, "'Semantic object classes in video: A high-definition ground truth database," *Pattern Recog.Lett.*, vol. 30, no. 2, pp. 88-97.

[66] S. Chougule, A. Ismail, A. Soni, N. Kozonek, V. Narayan and M. Schulze, "'An efficient encoder-decoder CNN architecture for reliable multilane detection in real time," *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1444-1451.

[67] A. Paszke, A. Chaurasia, S. Kim and E. Culurciello, "'Enet: A deep neural network architecture for real-time semantic segmentation," *arXiv preprint arXiv:1606.02147*.

[68] J. Long, E. Shelhamer and T. Darrell, "'Fully convolutional networks for semantic segmentation,", pp. 3431-3440.

[69] D. Teso-Fz-Betono, E. Zulueta, A. Sanchez-Chica, U. Fernandez-Gamiz and A. Saenz-Aguirre, "'Semantic Segmentation to Develop an Indoor Navigation System for an Autonomous Mobile Robot," *Mathematics*, vol. 8, no. 5, MAY, pp. 855.

[70] G. Csurka, D. Larlus, F. Perronnin and F. Meylan, "'What is a good evaluation measure for semantic segmentation?.", vol. 27, no. 2013, pp. 10.5244.

# Anexo I: Pliego de condiciones

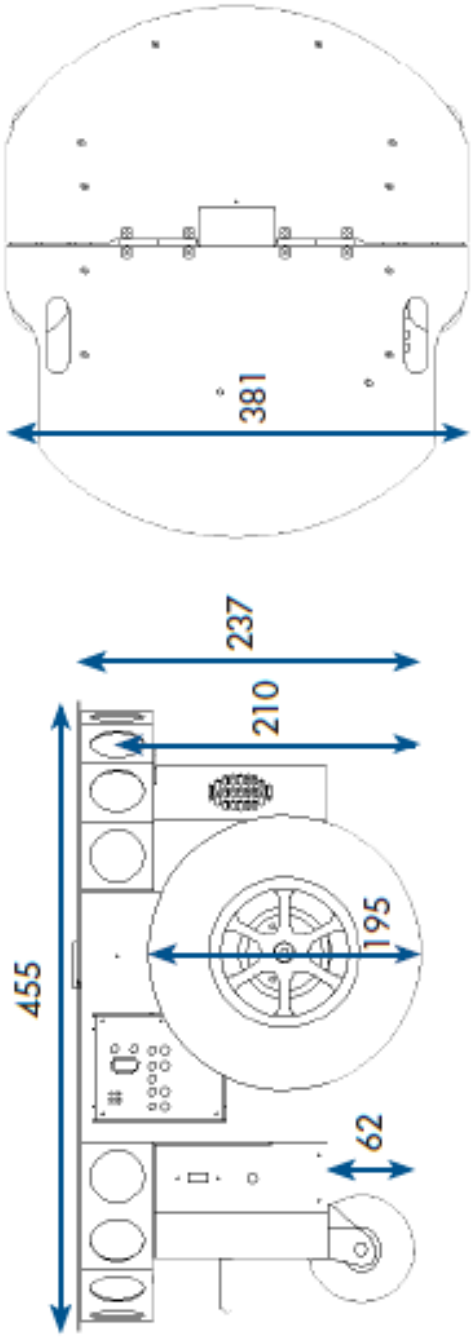
Se permite a la UPV-EHU, al departamento de Ingeniería de Sistemas y Automática, y al tutor del trabajo de fin de Master, Ekaitz Zulueta publicar y modificar la presente memoria.

Los archivos de Matlab y Arduino necesarios para la ejecución del algoritmo de navegación, descritos en el Anexo siguiente, son accesibles a través del siguiente enlace:

Contraseña: TFMAnder

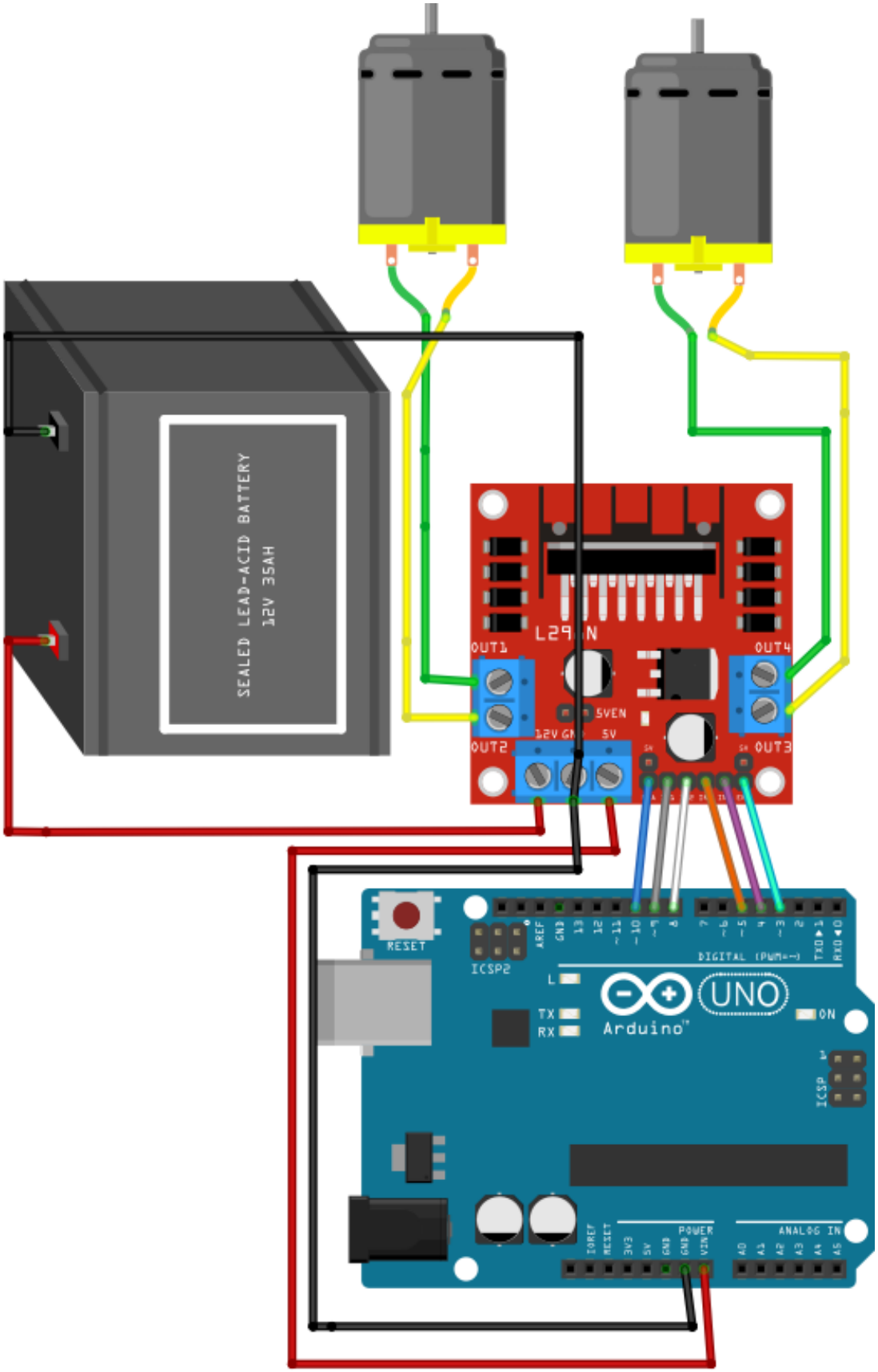
# Anexo II: Planos, Esquemas y Código

## Dimensiones del robot Pioneer 3-DX




Unidades en cm.

## Esquema del rediseño de hardware basado en la placa Arduino Uno



## Código de control de Motores en Arduino

```
/*         Control de Motores por puerto serie del Pioneer 3 V2.0
 *
 * Este código permite recibir las consignas de velocidad por puerto
*serie. Dichas Consignas son transformadas
 * En valores numéricos. Estos valores se corresponden con el valor de
salida del PWM que actúa sobre cada Motor
 *
 * El programa está diseñado para ejecutarse en una placa de Arduino
UNO
 *
 * Versión 2.0: Diseño de la trama corregido para reducir el tiempo
consumido en la comunicación en el programa principal de
 * navegación.
 * Ahora al enviar una consigna de velocidad se envía una única trama
en la que se buscan los identificadores de las consignas
 * de dutty para el PWM de cada rueda 'R' y 'L'.
 *
 * El programa permite enviar comandos específicos de parada 'S' y
arranque 'G'.
 * Las consignas de velocidad se deben encabezar por un identificador
'V'.
 * Ejemplo de trama para una consigna de dutty del PWM de 100 en la
rueda derecha y 80 en la rueda izquierda:
 *
 *    'VL80R100'
 */

/* Definición de Pines */
// Motor A
#define  ENA  10
#define  IN2  8
#define  IN1  9

// Motor B
#define  ENB  3
#define  IN3  4
#define  IN4  5


// Debug
#define Led_Pin 13


/* Definicion de variables */
volatile bool serial_flag = false;
volatile String inputString = "";
String copyString = "";
String velizq = "";
String velder = "";
char Label = "";
int deridx;

volatile int Vel_Izq = 0;
volatile int Vel_Der = 0;


void setup() {
/* Configuracion del Puerto Serie */
  Serial.begin(19200);
```

```
  while (!Serial) {
    ; // Esperar a que se establezca la comunicación serie
  }
  Serial.setTimeout(25);

  Serial.flush();

  Serial.println("Comunicación Iniciada Puerto COM4");

  delay(100);

}

void loop() {
/*Comprobar si se ha recibido una nueva trama */
  if (serial_flag) {
    serial_flag = false;  // Devolver a false para no ejecutarla
continuamente despues de la 1a trama
    Label = inputString.charAt(0);
    /* Si se ha detectado nueva trama se lee el identificador (1er
elemento de la trama) y se realiza la acción asociada*/
    switch (Label) {
      case 'V': //"Velocidad" Asignar consigna del PWM a los Motores
de las Ruedas
        velizq = inputString.substring(2); // El índice 1 se
corresponde con el identificador 'L'
        Vel_Izq = velizq.toInt()  ;
        deridx = inputString.lastIndexOf('R');  // Búsqueda del
identificador 'R' en la trama
        velder = inputString.substring(deridx + 1); // La consigna se
encuentra en la siguiente posicion de la trama
        Vel_Der = velder.toInt()  ;
        break;
      case 'S': //"Stop" Detener los motores
        Vel_Izq = 0;
        Vel_Der = 0;
        break;
      case 'G': //"Go" Arrancar ambos motores a baja velocidad
        Vel_Izq = 50;
        Vel_Der = 50;
        break;
      case 'T': // "Test Message" para depuration
        ;
        break;
      default:
        Vel_Izq = 0;
        Vel_Der = 0;
        break;
    }
    /*Asignar las nuevas consignas a los PWM*/
    noInterrupts();
    set_vel_forward(Vel_Der, Vel_Izq);
    interrupts();
  }

}
/*Subfuncion que asigna las consignas del PWM a los pines adecuados*/
void set_vel_forward(int vel_A, int vel_B) {
  // vel_A --> Rueda Derecha
  // vel_B --> Rueda Izquierda
  /*Comprobación de seguridad*/
```

```
  if (vel_A > 255 or vel_B > 255) {
    analogWrite (ENA, 0);
    analogWrite (ENB, 0);
    Serial.println("Dutty Cicle execedido!!!!");
    return;
  }

  //Direccion motor A (derecho)
  digitalWrite (IN1, HIGH);
  digitalWrite (IN2, LOW);
  analogWrite (ENA, vel_A); //Velocidad motor A
  //Direccion motor B (izquierdo)
  digitalWrite (IN3, LOW);
  digitalWrite (IN4, HIGH);
  analogWrite (ENB, vel_B); //Velocidad motor B

}
/*Interrupción del puerto serie
* Al finalizar la ejecución de un loop principal, si ha llegado una
nueva trama, salta la interrupción y se ejecuta la
* función seralEvent()
*/
void serialEvent() {

  inputString = Serial.readString();  //Almacenar la trama en una
string volatile
  copyString = inputString.substring(0); // Ñapa para castear volatile
S a string

  serial_flag = true; // Activar para ejecutar la sección
correspondiente del código principal

}
//Nota: las RSI deben de ser lo más breves posibles, por eso se delega
el tratamiento de la trama al código principal
//Mediante la variable serial_flag
```

## Código para el procesamiento de Imagen y Navegación en Matlab

### Programa principal: segmentation_in_the_loop_Line_following_Telemetria_V3.m

```
%               Segmentation in the loop for line Following
clearvars -except net netinfo
close all
addpath('Imagenes Ejemplo')
clc
% Descripcion del programa:
% Realizar un script que tome imagenes de la cámara, las
segmente,traze una
% ruta y envie una consigna mediante puerto serie.
% Medir el tiempo de ejecucion del programa

%% Configuracion de la Cámara
divice_number=2;
cam = webcam(divice_number);
cam.Resolution='640x360'; % Mismo Tamaño que la capa de entrada de la
red

%   Imagen de prueba
Im_original=snapshot(cam);
Im_original_size=size(Im_original,(1:2));
%   Es posible editarlos.
%   Device Specific Properties:
%         BacklightCompensation = on
%         Brightness = 128
%         Contrast = 128
%         FrameRate = 30.0000
%         Saturation = 128
%         Sharpness = 128
%         WhiteBalance = 417
%         WhiteBalanceMode = auto
%   Ejemplo: src.Brightness=100;

%% Cargar red Neuronal
if ~exist('net','var')
%     load ('Line_Detector_Resnet_18')
     load('E_D_CNN_Trained.mat')

end
net_input_size=net.Layers(1).InputSize(1:2);
cmap = ade20kColorMap2_Lineas;
net_size_Res=[num2str(net_input_size(2)),'x',num2str(net_input_size(1)
)];
if ~strcmp(net_size_Res,cam.Resolution)
    disp('Resolucion de la cámara o tamaño de la capa de entrada
erronea')
    return
end
%% Configurar comunicacion serie
%  La comunicacion se realiza a traves de los puertos COM 4 y 5, donde
esta
%  conectados el convertidor serie-usb y el arduino.

%  En el arduino hay un programa ECHO. Se le envia la consigna de vel
de
%  las ruedas y el arduino devuelve esa informacion por el otro puerto
para
%  poder monitorear la comunicacion
```

```
baudrate=19200;
Arduino="COM3";
% Arduino="COM4";
serial=serialport(Arduino,baudrate);
configureTerminator(serial,0);
pause(1)
writeline(serial,'##############################');
writeline(serial,'Inicio de la comunicacion serie');
%% Configurar ventanas de graficos
% figure(1)
% movegui(1,'northwest')
% figure(2)
% movegui(2,'southeast')
%% Dummy Segmentation
% load('prueba')
Im=imread('Ejemplo_pasillo_4.bmp');
Im_reducida=imresize(Im,net_input_size);
C_reducida = semanticseg(Im_reducida, net);
%% Loop
run=true;
count=1;
mostrar=false;
execution_time_array=zeros(300,1);
i=1;
Test_limit=500;
Telemetria.Velocidad.Izquierda=zeros(Test_limit,1);
Telemetria.Velocidad.Derecha=zeros(Test_limit,1);
Telemetria.Ptos_Control=zeros(Test_limit,2);
Telemetria.Tiempo_Ejecucion=zeros(Test_limit,1);
Telemetria.Tiempo_Segmentacion=zeros(Test_limit,1);
Telemetria.Tiempo_calcularVel=zeros(Test_limit,1);
Telemetria.Tiempo_Adquirir_Imagen=zeros(Test_limit,1);
Telemetria.Tiempo_ptsMedios=zeros(Test_limit,1);
Telemetria.Tiempo_ptsControl=zeros(Test_limit,1);
Telemetria.Tiempo_enviarVel=zeros(Test_limit,1);
main_loop_tic=tic;
while(run)

    %% Adquirir imagen de la cámara
    adquirir_tic=tic;
    Im=snapshot(cam);
    Telemetria.Tiempo_Adquirir_Imagen(count)=toc(adquirir_tic);
%     Im=imread('Ejemplo_pasillo_2.bmp'); % Prueba para depuracion

%% Segmentacion
    % Resize de la imagen
%     Im_reducida=imresize(Im,net_input_size);
    % Segmentar imagen
    segmentation_tic=tic;
    C=semanticseg(Im, net);
    Telemetria.Tiempo_Segmentacion(count)=toc(segmentation_tic);

    if mostrar
        figure(1)
        imshow(Im)
        hold on
        title ('Cámara')
    end

    if mostrar
        figure(2)
```

```
        B = labeloverlay(Im,C,'Colormap',cmap,'Transparency',0.4);
        imshow(B)
        hold on
        title ('Segmentado')
    end
    %% Postprocesado
    % No se realiza de momento

    %% Calculo de pts medios
    pts_medios_tic=tic;
    [Ptos_medios,Limit_points] = Calcular_ptos_medios(C);
    Telemetria.Tiempo_ptsMedios(count)=toc(pts_medios_tic);

    %% Calculo del Pto de Control
    pts_control_tic=tic;
    [Pto_Control] =
calcular_pto_control(Ptos_medios,Im_original_size);
    Telemetria.Ptos_Control(count,:)=Pto_Control;
    Telemetria.Tiempo_ptsControl(count)=toc(pts_control_tic);
    if mostrar
        figure(1)
        hold on
        plot(Limit_points(:,1),1:size(C),'xr');  %Extremos iniciales
        plot(Limit_points(:,2),1:size(C),'xb');  %Extremos finales
        plot(Ptos_medios(:,1),1:size(C,1),'.g')  %Ruta optima
        plot(Pto_Control(2),Pto_Control(1),'om','Markersize',15)
        plot(1:net_input_size(2),1/2*net_input_size(1)*...
             ones(1,net_input_size(2)),'.m') % Limite inferior de
Control
        plot(1:net_input_size(2),net_input_size(1)*...
             ones(1,net_input_size(2)),'.m') % Limite superior de
Control
        title ('Cámara')
        legend('Extremo izquierdo','Extremo derecho','Trayectoria',...
               'Pto Control','Zona Control')
    end
    %% Calculo de vel de ruedas
    % Ecc de Dani
    calcularVel_tic=tic;
    [Vel_izq,Vel_der] = Calcular_vel(Pto_Control,Im_original_size);
    Telemetria.Tiempo_calcularVel(count)=toc(calcularVel_tic);
%         display('##############################')
%         display(['Vel izquierda: ',num2str(Vel_izq)])
%         display(['Vel derecha: ',num2str(Vel_der)])
    %% Enviar consigna a ruedas por puerto serie
    enviarVel=tic;
    send_dutty_V2(serial,Vel_izq,Vel_der);
    Telemetria.Tiempo_enviarVel(count)=toc(enviarVel);
    Telemetria.Velocidad.Izquierda(count)=Vel_izq;
    Telemetria.Velocidad.Derecha(count)=Vel_der;

    % Aumentar contador para medir tiempo de ejecucion
    count=count+1;
    % Comprobar condicion deparada
    if count==Test_limit
        disp('############### Time Out ############')
        run=false;
        pause(0.0035)
        writeline(serial,'S');
        send_dutty(serial,0,0);
        break
```

```
    end

    pause(0.0035)
%     writeline(serial,num2str(i));
Telemetria.Tiempo_Ejecucion(count)=toc(main_loop_tic);
end
writeline(serial,'S');
pause(0.0035)
send_dutty(serial,0,0);
```

### Subfunción 1: Calcular_pto_control.m

```
      function [Pto_Control] =
      calcular_pto_control(Ptos_medios,Im_size)%#codegen
          Pto_Control=[];
          %   Eje central
          eje=ones(Im_size(1),1)*Im_size(2)/2';
          suelo_no_detectado=Ptos_medios==0;
          %   EL rango de calculo del pto

          Puntos_limite=(Ptos_medios-eje);
          Puntos_limite(suelo_no_detectado)=0;

          [~,
      ref]=max(abs(Puntos_limite(round(1/2*Im_size(1)):Im_size(1))));

          Pto_Control=[round(1/2*Im_size(1))+ref-1
      ,Ptos_medios(round(1/2*Im_size(1))+ref-1)];


      end
```

### Subfunción 2: Calcular_ptos_medios.m

```
      function [Pto_Control] =
      calcular_pto_control(Ptos_medios,Im_size)%#codegen
          Pto_Control=[];
          %   Eje central
          eje=ones(Im_size(1),1)*Im_size(2)/2';
          suelo_no_detectado=Ptos_medios==0;
          %   EL rango de calculo del pto

          Puntos_limite=(Ptos_medios-eje);
          Puntos_limite(suelo_no_detectado)=0;

          [~,
      ref]=max(abs(Puntos_limite(round(1/2*Im_size(1)):Im_size(1))));

          Pto_Control=[round(1/2*Im_size(1))+ref-1
      ,Ptos_medios(round(1/2*Im_size(1))+ref-1)];

      end
```

### Subfunción 3: Calcular_Vel

```
function [Vel_izq,Vel_der] =
Calcular_vel(Pto_Control,Im_size)%#codegen
Vel_nom=0.15;
Vel_min=0.025;
Vel_der=[];
```

```
Vel_izq=[];
eje_central=Im_size(2)/2;
    Dist=eje_central-Pto_Control(2);
distancia_max=640;
compensacion_maxima=0.5;

compensacion=min(abs(Dist),distancia_max)/distancia_max;
% compensacion=max(compensacion,compensacion_maxima);
    if sign(Dist)<0
        Vel_izq=Vel_nom;
        Vel_der=Vel_nom*(1-compensacion);
    elseif sign(Dist)> 0
        Vel_der=Vel_nom;
        Vel_izq=Vel_nom*(1-compensacion);
    end
  % Asegurar vel minima
%   Vel_der=max(Vel_der,Vel_min);
%   Vel_izq=max(Vel_izq,Vel_min);
  % Pasar al rango 0-255
  Vel_der=round(255*Vel_der);
  Vel_izq=round(255*Vel_izq);

End
```

Subfunción 4: Send_dutty_V2.m

```
function  send_dutty(serial,dutty_izq,dutty_der)
    %% Enviar por el puerto serie el dutty para el control de
las ruedas
%     pause(0.035)

writeline(serial,['V','L',num2str(dutty_izq),'R',num2str(dutty_d
er)]);

end
```

## Código para el entrenamiento de Redes Neuronales Convolucionales en Matlab

### Entrenamiento de la Red CNN

## Entrenamiento de CNN

```
clearvars
close all
clc
```

## Nuevas clases y etiquetas

Extraer las características del etiquetado realizado por Aitor.

2 clases:

- Línea
- No línea

Se extrae  el nombre, el id ()

```
 load gTruth_LineaAitor %
 % Definir clases
 class1=cell2mat(table2cell(gTruth_LinraAitor.LabelDefinitions(1,1)));
 class2=cell2mat(table2cell(gTruth_LinraAitor.LabelDefinitions(2,1)));
 classNames = [convertCharsToStrings(class1);
               convertCharsToStrings(class2)];
 % Definir etiquetas

 labelIDs =
{cell2mat(gTruth_LinraAitor.LabelDefinitions.PixelLabelID(1));

cell2mat(gTruth_LinraAitor.LabelDefinitions.PixelLabelID(2))};
```

## Crear Bases de datos

Definir las carpetas en las que se encuentran las imágenes y sus etiquetas. Es necesario cambiar el path entero para que funcione en tu ordenador.

Imágenes

```
 imgpath = 'C:\Users\ander\Documents\MATLAB\Pasillos con seguimiento de
Linea Redimensionado Prop original\imagenesPasillos';

 % imageDS = imageDatastore(fullfile(imgpath,'Images'));
 imageDS = imageDatastore(imgpath);
```

Etiquetas

```
 labelpath='C:\Users\ander\Documents\MATLAB\Pasillos con seguimiento de
Linea Redimensionado Prop original\Labels';
 % LabelDS =
pixelLabelDatastore(fullfile(imgpath,'Labels'),classNames,labelIDs);
 LabelDS = pixelLabelDatastore(labelpath,classNames,labelIDs);
```

## Comprobar Bases de Datos

Leer una de las imágenes de la base de datos y superponer el etiquetado.

```
 I = readimage(imageDS,1);
 imshow(I)
```

```
 C = readimage(LabelDS,1);
 cmap = ade20kColorMap2_Lineas; % Función para colorear las etiquetas
 B = labeloverlay(I,C,'ColorMap',cmap);
 imshow(B)
 pixelLabelColorbar(cmap,classNames);
```

## Análisis de la base de Datos

Cálculo de estadísticas y pesos necesarios para el entrenamiento.

```
tbl = countEachLabel(LabelDS);
numberPixels = sum(tbl.PixelCount);
frequency = tbl.PixelCount / numberPixels;
classWeights = median(frequency)./ frequency;
```

## Set up entrenamiento

Definir:

- Tamaño de las imágenes de la base de datos.
- Numero de clases

*** El tamaño de la imagen a clasificar es elección del diseñador. Se debe ajustar al tamaño de las imágenes de la base de datos si esta es homogénea**

```
imageSize=[360 640 3];
numClasses = numel(classNames);

% Cargar E-net
load('E_D_CNN_architecture_weights_transfers.mat') % Se carga lgraph en
el workspace
% Crear nueva capa de clasificación de píxeles
pxLayer = pixelClassificationLayer('Name', 'labels', 'Classes',
tbl.Name, 'ClassWeights', classWeights);
% Reemplazar capa clasificación
% E_D_CNN_architecture_weights_transfers =
replaceLayer(E_D_CNN_architecture_weights_transfers,"pixel-class",
pxLayer);
E_D_CNN_architecture_weights_transfers(end)=pxLayer;
% Almacen de datos con imágenes y etiquetas de entrenamiento
% pximdsTrain = pixelLabelImageDatastore(imageDS,LabelDS);
% pximdsTrain = pixelLabelImageDatastore(ResizedImageDS,ResizedLabelDS);
```

## Partición de la base de datos para entrenamiento y validación

Dividir la base de datos en entrenamiento y validación para evitar el sobreentrenamiento

```
[imdsTrain, imdsVal, pxdsTrain, pxdsVal] =
partitionData(imageDS,LabelDS,labelIDs);
pximdsTrain = pixelLabelImageDatastore(imdsTrain,pxdsTrain);
pximdsVal = pixelLabelImageDatastore(imdsVal,pxdsVal);
```

## Opciones entrenamiento

Ajustar opciones de entrenamiento

```
options = trainingOptions('sgdm', ...
'LearnRateSchedule','piecewise',...
'LearnRateDropPeriod',4,...
'LearnRateDropFactor',0.5,...
'Momentum',0.9, ...
'InitialLearnRate',0.01, ...
'L2Regularization',0.005, ...
'MaxEpochs',4, ...
'MiniBatchSize',1, ...
'ValidationFrequency',35, ...
"ValidationPatience",inf,...
'ValidationData',pximdsVal,...
'Shuffle','every-epoch', ...
'ExecutionEnvironment','gpu', ... % Buscar gráficas
'CheckpointPath', tempdir, ...
"Verbose",true,...
'VerboseFrequency',50,...
'Plots','training-progress');
%'ValidationData',CombinedValImageLabelDs,
% pause
```

## Entrenamiento

Inicializar la Gpu

```
gpuDevice(1);
```

Comenzar entrenamiento (en mi ordenador tardo unos de 35 minutos)

```
[net, netinfo] =
trainNetwork(pximdsTrain,E_D_CNN_architecture_weights_transfers,options)
```

## Guardar Red

Se genera un nombre automáticamente tomando la fecha y la hora a la que finaliza el entrenamiento.

Se guarda todo el workspace empleado en el entrenamiento en un .mat

```
date_and_time=datestr(now,'dd_mm_yyyy_HH_MM_SS');
Net_name= 'Trained_net';
save_name=strcat(Net_name,'_',date_and_time)
```

```
save_name = 'Trained_net_19_05_2021_13_53_10'
```

```
save(save_name)
```

## Funciones auxiliares

Esta función permite añadir la barra lateral que indica las clases al superponer el segmentado en una imagen.

```
function pixelLabelColorbar(cmap, classNames)
% Add a colorbar to the current axis. The colorbar is formatted
% to display the class names with the color.

colormap(gca,cmap)

% Add colorbar to current figure.
c = colorbar('peer', gca);

% Use class names for tick marks.
c.TickLabels = classNames;
numClasses = size(cmap,1);

% Center tick labels.
c.Ticks = 1/(numClasses*2):1/numClasses:1;

% Remove tick mark.
c.TickLength = 0;
end
```

Esta función permite dividir una base de datos con sus etiquetas en dos subconjuntos para entrenamiento y validación de forma aleatoria.

```
function [imdsTrain, imdsVal, pxdsTrain, pxdsVal] =
partitionData(imds,pxds,labelIDs)
% Partition CamVid data by randomly selecting 60% of the data for
training. The
% rest is used for testing.

% Set initial random state for example reproducibility.
rng(0);
numFiles = numel(imds.Files);
shuffledIndices = randperm(numFiles);

% Use 60% of the images for training.
numTrain = round(0.75 * numFiles);
trainingIdx = shuffledIndices(1:numTrain);

% Use 20% of the images for validation
numVal = round(0.25 * numFiles);
valIdx = shuffledIndices(numTrain+1:numTrain+numVal);

% Create image datastores for training and test.
trainingImages = imds.Files(trainingIdx);
```

```
valImages = imds.Files(valIdx);


imdsTrain = imageDatastore(trainingImages);
imdsVal = imageDatastore(valImages);


% Extract class and label IDs info.
classes = pxds.ClassNames;
% labelIDs = camvidPixelLabelIDs();

% Create pixel label datastores for training and test.
trainingLabels = pxds.Files(trainingIdx);
valLabels = pxds.Files(valIdx);

pxdsTrain = pixelLabelDatastore(trainingLabels, classes, labelIDs);
pxdsVal = pixelLabelDatastore(valLabels, classes, labelIDs);

end
```

Redimensionamiento de bases de datos

# Redimensionar el dataset de Pasillo

```
clearvars
close all
clc
```

## Obtener Listado de Imágenes y Segmentado para Entrenamiento

```
images='C:\Users\ander\OneDrive\Documents\MATLAB\Pasillos con
seguimiento de Linea\imagenesPasillos\*.bmp';
Image_list=dir(images);
labels='C:\Users\ander\OneDrive\Documents\MATLAB\Pasillos con
seguimiento de Linea\PixelLabelData\*.png';
Label_list=dir(labels);
```

## Definir nuevo tamaño de Imágenes

```
imageSize=[360 640];
```

## Redimensionamiento de imágenes para entrenamiento

Leer las imágenes del dataset original, redimensionarlas y guardarlas en una nueva carpeta

Crear carpeta para almacenar la nueva base de daos de imágenes

```
 proyect_folder='C:\Users\ander\OneDrive\Documents\MATLAB';
 proyect_name='Pasillos con seguimiento de Linea Redimensionado Prop
original';
 proyect_path=fullfile(proyect_folder,proyect_name);
 if ~exist(proyect_path,'dir')
     mkdir(proyect_path)
 end
```

Generar nuevas Imágenes para entrenamiento

```
 new_Image_folder='imagenesPasillos';
 new_Image_folder_path=fullfile(proyect_path,new_Image_folder);

 if ~exist(new_Image_folder_path,"dir")
     mkdir(new_Image_folder_path)
 end
 k=1;
 for i=1:length(Image_list)
     % Cargar imagen
     image=fullfile(Image_list(i).folder,Image_list(i).name);
     im=imread(image);
     % Redimensionar
     im=imresize(im,imageSize);

     % Guardar en nueva direccion

new_name_and_path=fullfile(new_Image_folder_path,Image_list(i).name);
     imwrite(im,new_name_and_path,'jpg')
 %       if size(im,3) == 3 && ~exist(new_name_and_path,"file")
 % %      [status, msg, msgID]= copyfile(image, new_Image_folder_path);
 %          imwrite(im,new_name_and_path,'jpg')
 %       elseif size(im,3) == 1 && exist(new_name_and_path,"file")
 %           delete  (new_name_and_path)
 %           bw_ref_list(k)=i;
 %           k=k+1;
 %       end
 end
```

Generar nuevas etiquetas para entrenamiento

```
 beep
 new_Label_folder= fullfile('training','Labels');
 new_Label_folder_path=fullfile(proyect_path,new_Label_folder);

 if ~exist(new_Label_folder_path,"dir")
     mkdir(new_Label_folder_path)
 end
 k=1;
 for i=1:length(Label_list)
```

```
    % Cargar etiqueta
    label=fullfile(Label_list(i).folder,Label_list(i).name);
    lb=imread(label);
    % Redimensionar etiqueta
    lb=imresize(lb,imageSize);
    % Guardar en nueva direccion
    %     [status, msg, msgID]= copyfile(label, new_Label_folder_path);

new_name_and_path=fullfile(new_Label_folder_path,Label_list(i).name);
    imwrite(lb,new_name_and_path,'png')
%     if i ~= bw_ref_list(k)
%         imwrite(lb,new_name_and_path,'png')
%     else
%         delete (new_name_and_path)
%         k=k+1;
%         if k >= 5
%             k=4;
%         end
%     end
end
```

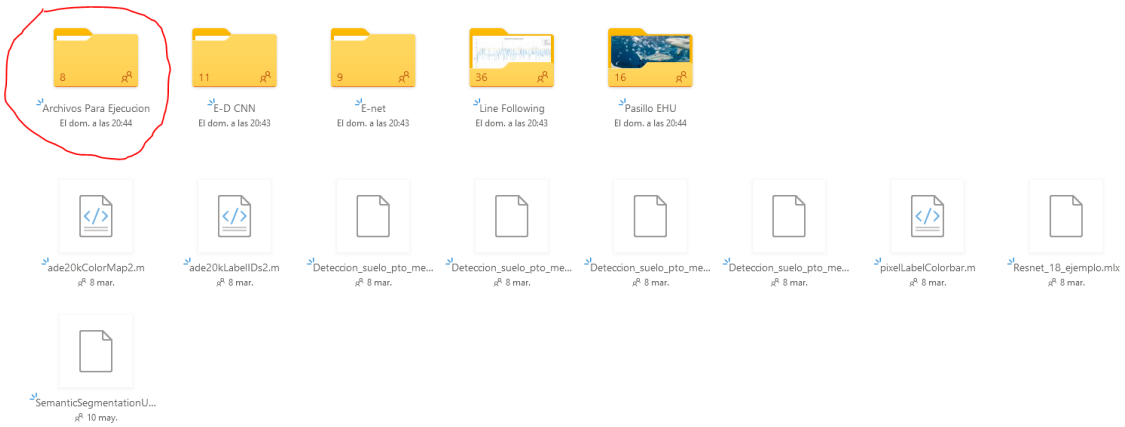
# Anexo III: Manuales de Usuario

Los archivos necesarios para la ejecución del algoritmo de control, así como otros códigos de interés, incluidos los presentados en el Anexo II: Planos, Esquemas y Código, son accesibles a través del enlace a la carpeta de OneDrive dispuesta en el Anexo I: Pliego de condiciones. La contraseña para poder acceder a dichos documentos también está presente.

En este anexo se especifica cuáles son los elementos de código mínimos para la puesta en marcha del algoritmo de navegación.

En la citada carpeta de OneDrive se presentan dos carpetas. La primera, referida al código de Arduino presenta un único proyecto con el código fuente a quemar en la placa. Este código está diseñado para ejecutarse en un Arduino UNO, pero es compatible con MEGA y otras afines.

La segunda carpeta está referida a los códigos desarrollados en Matlab. La siguiente imagen muestra la vista de dicha carpeta en el navegador. La carpeta marcada en rojo, contiene todos los archivos necesarios para ejecutar el algoritmo de navegación y localización.




Una vez descargado el contenido, se coloca el "path" de Matlab en dicha carpeta. El script a ejecutar es "segmentation_in_the_loop_Line_following_Telemetria_V3".

Para el correcto funcionamiento del algoritmo es necesario conectar el Arduino al PC mediante el cable USB al igual que la cámara web. Es posible que el identificador de la cámara o del canal de comunicación serie muestre un error debido a que cada PC asigna uno diferente.

En general la webcam es asignada al "device_number" 2, ya que la mayoría de los portátiles poseen una cámara integrada. Sin embargo, hay ocasiones en las que Matlab les asigna el orden contrario.

Por otro lado, al configurar la comunicación serie es necesario indicar el número del puerto. El número se asigna empleando la estructura "COMX" donde 'X' es el número de puerto asignado. Este número puede comprobarse mediante el IDE de Arduino o a través del administrador de dispositivos de Windows.

Cualquier duda, ante posibles fallos del algoritmo, pueden comunicarse a través del correo de contacto: ander.sanchez@ehu.eus