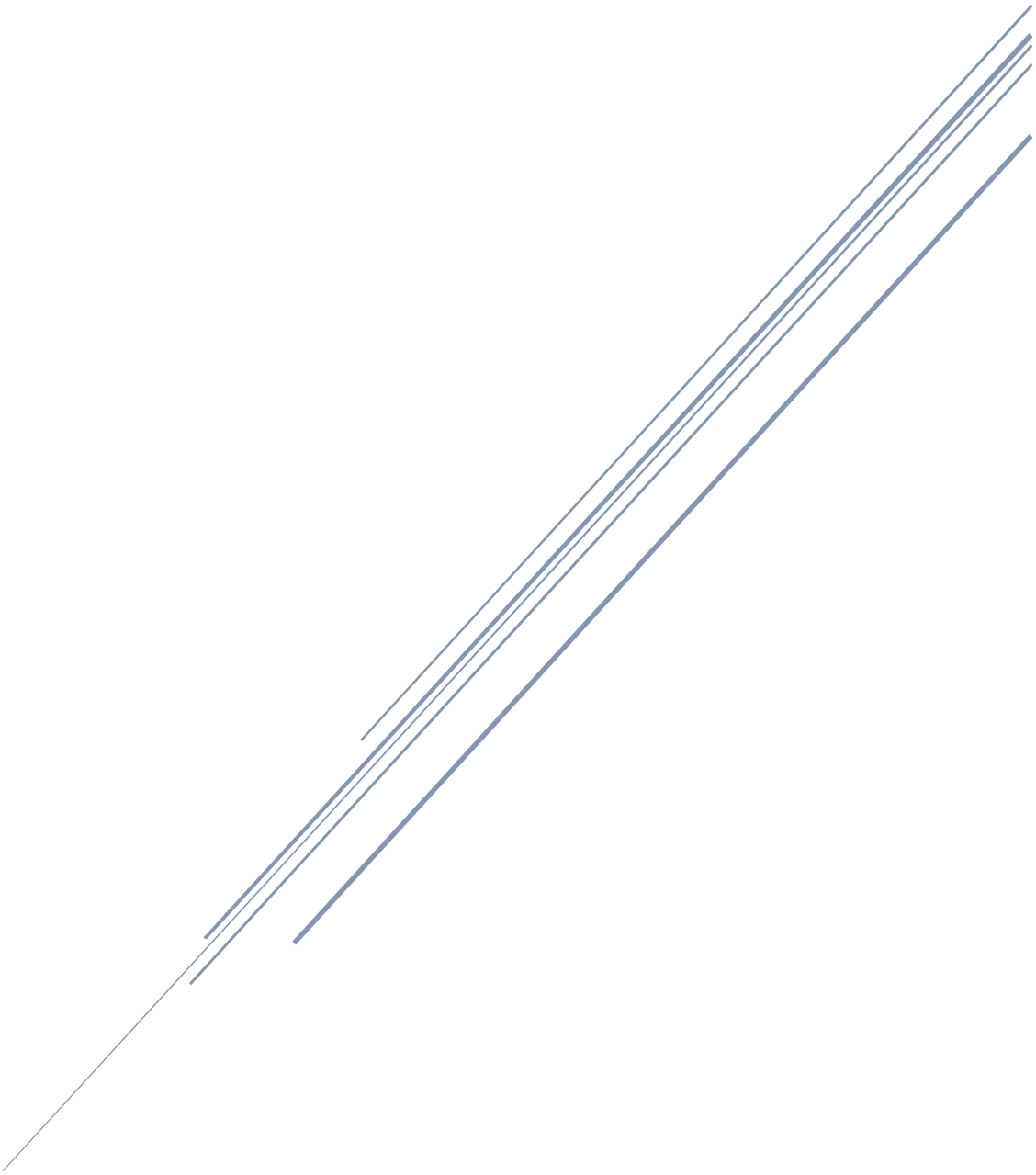
TRABALLO FINAL DE MATERIA

Pablo Parada Souto
Daniel Castro Gómez



EPSE Lugo
Visión Artificial Avanzada

Índice

Introducción

En este documento vamos a explicar, de una forma detallada y concisa, todas las partes en las que hemos decidido dividir nuestro proyecto final de la materia de Visión Artificial Avanzada. También comentaremos los resultados que obtuvimos durante las distintas pruebas que realizamos durante la realización de este trabajo y, por último, comentaremos las conclusiones que obtenidas.

La plataforma de nuestro trabajo será el uso del robot móvil de la marca Waveshare, en concreto el modelo JetRacer ROS AI. Este robot cuenta con cuatro ruedas, 2 delanteras direccionables y 2 traseras de propulsión. La tecnología con la que cuenta este robot principalmente es una NVIDIA Jetson Nano, la cual nos ayudó para poder ejecutar tecnologías, pero comentaremos en siguientes apartados como nos fue útil el uso de esta placa. El robot está controlado mediante ROS, el cual nos permite tener un control de todos los actuadores y de sensores con los que cuenta. En esta ocasión los sensores que vamos a usar es una cámara monocular y un sensor láser, en concreto un sensor RPLIDAR.

El objetivo de nuestro trabajo es la creación de un sistema de conducción autónoma. Posteriormente vamos a implementarlo en el robot que hemos comentado anteriormente.

En esta ocasión hemos decidido dividir nuestro trabajo en tres partes diferenciadas, las cuales tienen un objetivo diferente. Una de ellas es la que se interconecta con las otras dos mediante el uso de ROS. Después, una de las partes se encarga de la detección de los objetos mediante el uso de técnicas de visión por computadora y la restante se encarga del movimiento del robot.

Las técnicas de visión por computadora están basadas en redes neuronales las cuales están entrenadas para poder obtener características de una imagen o de un video, ya que un video son una serie de imágenes seguidas a un cierto tiempo. Este tipo de técnicas de visión por computadora nos permiten la realización de un gran número de posibilidades, como la clasificación, segmentación o detectar objetos.

El uso de redes neuronales en técnicas de visión por computadora cuenta con numerosas ventajas ante el uso de técnicas de visión clásica. Una serie de ventajas son las siguientes: capacidad de aprendizaje automático, mejor capacidad de generalización, flexibilidad y adaptación… Estas ventajas nos permiten obtener un rendimiento superior en un gran número de tareas, lo que las hace una opción atractiva y poderosa para una amplia gama de aplicaciones.

Por último, puesto que nos encontramos usando un robot a escala, la cámara capta la información desde un punto en el cual en los escenarios reales no se cumplen, es decir, no es lo mismo captar las imágenes desde un vehículo autónomo que desde el robot que estamos usando. Como consecuencia, es necesario generar una base de datos propia para después etiquetar, con softwares específicos, toda la información de dichas imágenes. A continuación, vamos a transferir esa información al detector de objetos para después ponerlo en producción.

Aclaraciones previas

NVIDEA Jetson Nano

La NVIDEA Jetson Nano es una plataforma de computación de inteligencia artificial (IA) diseñada específicamente para aplicaciones de aprendizaje profundo y visión por computadora en dispositivos embebidos y de bajo consumo de energía.

La placa esta equipa con un procesador ARM de cuatro núcleos y una GPU NVIDIA Maxwell con 128 núcleos CUDA y 4GB de memoria RAM. También cuenta con varios puertos de entrada y salida. Esto lo hace versátil y adecuado para una variedad de aplicaciones.

Las Jetson Nano son ampliamente utilizadas para aplicaciones de inteligencia artificial (IA) debido a varáis características que las hacen especialmente adecuadas para este propósito:

1. **Potencia de calculo**

Está equipada con potentes procesadores y unidades de procesamiento grafico (GPU) diseñadas específicamente para cargar de trabajo de IA. Estas GPU pueden realizar operaciones de cálculo en paralelo de manera muy eficiente, lo que las hace ideales para ejecutar algoritmos de aprendizaje profundo y redes neuronales

2. **Optimización de software**

NVIDEA proporciona herramientas y bibliotecas se software optimizadas para aprovechar al máximo el hardware. Esto incluye bibliotecas de cálculo numérico aceleradas por GPU, como cuDNN y TensorRT y CUDA Toolkit.

3. **Eficiencia energética**
4. **Soporte para frameworks de IA**
5. **Flexibilidad**

En resumen, las Jetson Nano son una opción popular para aplicaciones de IA por dichas características e ideales para una amplia gama de aplicaciones en el ámbito de la inteligencia artificial y la robótica.

Sistema Operativo Robótico (ROS)

Es un conjunto de herramientas y bibliotecas de software y bibliotecas de software de código abierto diseñados para ayudar a los desarrolladores de robots a crear software robusto y modular para una amplia variedad de aplicaciones robóticas.

Algunas características importantes de ROS incluyen:

1. **Arquitectura distribuida**

Está diseñado para sistemas robóticos distribuidos, lo que permite que los diferentes componentes del robot se comunican entre sí de manera eficiente a través de mensajes.

2. **Gestión de paquetes**

Los usuarios pueden compartir y reutilizar paquetes de software fácilmente a través del repositorio oficial de ROS.

3. **Herramientas de desarrollo**

Proporciona una variedad de herramientas de desarrollo incluyendo utilidades de línea de comandos para la depuración y administración del sistema

4. **Soporte de simulación**
5. **Compatibilidad con múltiples plataformas**

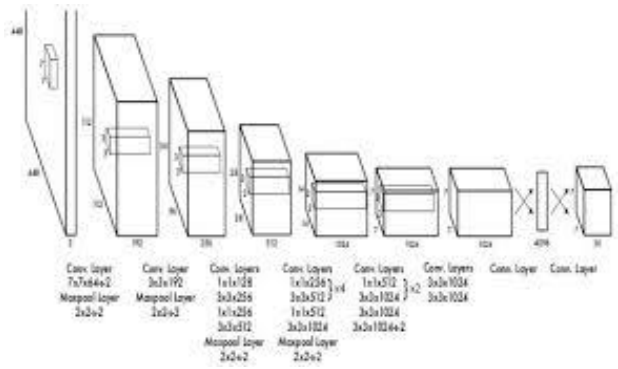
Es compatible con una amplia gama de plataformas de hardware y sistemas operativos, lo que permite a los desarrolladores trabajar con una variedad de robots y dispositivos.

En resumen, ROS es una plataforma flexible y potente que facilita el desarrollo de aplicaciones robóticas al proporcionar herramientas y bibliotecas que abordan muchos de los desafíos comunes en el desarrollo de sistemas.

Detector YOLO

El detector YOLO (You Only Look Once) es un modelo de detección de objetos en imágenes y videos en tiempo real. En esta ocasión estamos usando el modelo YOLO8 que corresponde a la versión 8 de este modelo, que comentaremos los avances que trae esta versión.

La arquitectura básica del modelo Yolo es el siguiente:



Este modelo para poder hacer la predicción y la clasificación de los objetos se realizan en los siguientes pasos:

1. **División de la imagen**

Se divide la imagen de entrada en una cuadricula generalmente en tamaño 7x7 o 9x9, dependiendo de la versión del modelo. Cada celda de la cuadricula se encarga de predecir la presencia y las características de los objetos dentro de esa región.

2. **Predicción de cajas delimitadores y probabilidades de clase**

En cada celda de la cuadricula, se predice múltiples cajas delimitadores que rodean los objetos, Para cada caja se predice la probabilidad de que esa caja contenga un objeto y la probabilidad de que ese objeto pertenezca a una a una clase específica.

3. **Regresión de coordenadas**

En lugar de predecir las coordenadas absolutas de las cajas delimitadoras, se predice desplazamientos relativos a la posición de la celda de la cuadrícula. Esto hace que el modelo sea más robusto a las variaciones en la posición y el tamaño de los objetos.

4. **Función de perdida**

Se utiliza una función de perdida que penaliza las predicciones incorrectas de las cajas delimitadoras y las clases. Esta función de perdida tiene en cuenta tato la precisión espacial de las cajas delimitadoras como la precisión de la clasificación

5. **Postprocesamiento y supresión de no máximos**

Después de todo se realiza un proceso de postprocesamiento para eliminar las detecciones redundantes y mantener solo las detecciones más confiables. Esto incluye supresión y no máximos, donde se eliminan las detecciones que tienen alta superposición con detección más confiables

6. **Salida final**

La salida del modelo es una lista de cajas delimitadoras junto con las clases predichas y las probabilidades de confianza correspondiente. Esto proporciona información sobre la ubicación y la clase de los objetos detectados en la imagen.

Como hemos visto en esta ocasión estamos utilizando la versión 8 de este modelo. Este modelo nos aporta la funcionalidad principal que es la detección y la clasificación de objetos mejorada y con más clases de objetos, con lo que nos permite obtener mejores resultados.

También nos aporta novedades como la segmentación de imágenes y la estimación de poses, en esta ocasión para poder simplificar el trabajo y debido a unas incompatibilidades con entre versiones en el robot real, hemos decido usar una de estas novedades para nuestro trabajo en esta ocasión hemos decidido usar la segmentación de imágenes para poder detectar las líneas de los carriles en los cuales nos movemos.

Detector YOLOv8 para segmentar

Como hemos comentado anteriormente Yolov8 es un algoritmo basado en redes neuronales convolucionales, que en estas nuevas versiones cuenta con la capacidad de segmentar imágenes.

La segmentación de imágenes es el proceso de identificar y delinear de forma precisa cada objeto detectado en la imagen. Para que podamos hacer esto con un modelo de Yolov8 necesitamos elegir una versión especifica con la cual cuenta con esta funcionalidad, esto se hace para que el modelo sea más eficiente.

La arquitectura es muy similar a la arquitectura del Yolo para detección, pero en esta ocasión el modelo cuenta con una cabeza para poder realizar dicha segmentación, pudiendo crear mascaras.

Los ventajes que tenemos al usar este algoritmo son las siguiente:

- **Velocidad**: Mantiene la eficiencia en tiempo real característica de los Yolo, lo que hace adecuado para aplicaciones donde la latencia es crítica.
- **Precisión:** Las mejoras en la arquitectura nos permiten una mayor precisión a la hora de detectar como la segmentación
- **Flexibilidad:** Capaz de manejar tanto la detección de objeto como la segmentación de manera eficiente

Desarrollo

En este apartado vamos a comentar las diferentes partes que hemos desarrollado para poder realizar el comportamiento que deseábamos.

Implementación de ROS (Código Main)

Una de las partes esenciales que necesitamos desarrollar es la parte de ROS, con las cuales podemos hacer todas las implementaciones necesarias para que todas las tecnologías funcionen.

Necesitamos hacer esta implementación para poder acceder a los sensores, los cuales hemos comentado anteriormente, y también necesitamos enviar información a los actuadores para que podamos realizar el comportamiento que deseamos

La implementación principal es crear el nodo principal, el cual se usa para ejecutar ros y poder implementar todas las funcionalidades necesarias. El código que usamos es el siguiente:

Acceder a la información de los sensores

Para poder acceder a la información que nos envían los sensores que vamos a usar es necesario crear un suscriptor a los topics de dichos sensores. Estos topic se encuentran enviando mensajes continuamente con los cueles podemos acceder a la información.

En esta ocasión los topic y tipo de mensaje correspondiente son los siguientes:

- **Sensor Laser**

Topic: “ /scan”

Mensaje: sensor_msgs/LaserScan [`float32 angle_min,float32 angle_max, float32 angle_increment, float32 time_increment, float32 scan_time,float32 range_min, float32 range_max, float32[] ranges, float32[] intensities`]

- **Cámara**

Topic: “ /camera/image_raw”

Mensaje: sensor_msgs/Image [`Header , uint32 height, uint32 width, string encoding, uint8 is_bigendian, uint32 step, uint8[] data`]

Enviar información a los actuadores

Para poder enviar información a los actuadores para poder realizar acciones, solamente es necesario crear los mensajes correspondientes y envíalos mediante un publicador el cual envía el mensaje correspondiente.

En esta ocasión hacemos tres publicadores, los cueles uno se encarga de los motores y los otros dos se encargan de enviar varias imágenes uno de ellos se encarga de enviar la imagen captada por la cámara y anotando las detecciones que detecta el modelo YOLO y el otro publicador envía la imagen de las máscaras resultantes del modelo YOLO entrenado para segmentación.

- **Motores**

Topic: “'/cmd_vel”

Mensaje: geometry_msgs/Twist [`Vector3 linear,Vector3 angular`]

- **Visualizador de detecciones**

Topic:” /ImagenDetecciones”

Mensaje: sensor_msgs/Image

- **Visualizador de mascaras**

Topic: “/ImagenMascaras”

Mensaje: sensor_msgs/Image

Divisiones del código
En esta ocasión, hemos decido dividir el código en tres partes diferenciadas, una que será el código principal y otras dos que son dos servidores a los cuales le vamos a enviar la información que necesitan, ellos harán la funcionalidad que le hemos asignado y nos responden con la información que le hemos definido.

Para poder hacerlo es necesario usar una utilidad que tiene ROS, que es la posibilidad de crear mensajes personalizados para poder enviar e información distinta al servidor. De esta manera podemos enviar toda la información necesaria para realizar las distintas funciones.

- **Main**

El main es el que se encarga de recoger los datos de los sensores e interconecta los diferentes servidores, enviándoles y recibiendo la información necesaria para poder realizar el comportamiento final.

- **Servidor de Analizar Datos**

Este servidor recibe toda la información de los sensores, que es captada en el main y va a analizar estos datos que necesitamos. Pero esto lo comentaremos más detalladamente en siguientes apartados.

- **Servidor de conducción**

Este servidor recibe la información que le envía el main con los datos que ha obtenido del servidor de analizar datos. Después, analiza la información obtenida para intentar mantenerse dentro del carril, pero lo comentaremos más detalladamente en siguientes apartados.

Servicio de analizar datos

Esta parte del código se encarga de coger toda la información que es captada en el apartado anterior y analizarla para poder obtener datos que son esenciales para poder conocer el entorno que nos rodea y después actuar respecto a dichos datos.

Las principales tareas que tiene que realizar este código es la detección de objetos dentro del rango de visión de la cámara y correlacionarla con la medida de distancia del sensor laser y también es la encargada de detectar las líneas de los carriles por los cuales circula el robot.

Para poder realizar estas tareas hemos creado un servicio de ros, al cual le llega la información que capta el programa main y le envía la información que ha generado de vuelta. La creación de este servicio nos permite independizar los códigos y que sea bastante modular a la hora de la

producción debido que son códigos totalmente independientes, solamente es necesario adaptar los datos de entrada.

Ahora vamos a comentar como hacemos la detección de los objetos y la segmentación. Primeramente, tenemos que cargar los modelos Yolov8, en este proyecto vamos a usar dos. El primero de ellos para la detección de objetos, en esta ocasión estamos usando el modelo preentrenado, de esta forma tenemos un gran número de clases con lo que nos permite abarcar bastantes posibilidades.

Después, para poder hacer la segmentación de las líneas hemos tenido que entrenar otro modelo, de esta manera nos aseguramos de que el modelo pueda segmentar correctamente las líneas. Para poder entrenarlo hemos desarrollado un código el cual carga un dataset de imágenes, las cuales etiquetamos gracias a la plataforma de Roboflow. Después, hemos iniciado el entrenamiento, los resultados del entrenamiento lo comentaremos en el apartado de resultados.

El servicio cuando se le envía una petición el ejecuta la función denominada *AnalizarDatosFuction(),* la propia función se encarga de recoger la imagen y los datos del láser.

Primero se recorta los datos del láser, para solamente tener los datos centrales, debido que son los datos que corresponden con el ángulo de visión de la cámara monocular que tenemos en el frente del robot.

Para el análisis de la imagen, este lo hacemos en dos pasos diferentes debido que son dos pasos totalmente independientes.

Detección de los objetos

Primeramente, vamos a hacer la detección de los objetos, para eso vamos a usar el modelo neuronal y obtenemos el objeto boxes, donde se encuentran todas las detecciones que ha hecho el modelo. Extraemos toda la información que necesitamos, como las posiciones, confianzas y clases. Después vamos a correlacionar las detecciones con las medidas del láser.

Para correlacionar las medidas con las detecciones, vamos a realizar los siguientes pasos:

- Primeramente, calculamos el rango de medidas que corresponde a la posición de la cámara
- Calculamos la media de las distancias, para poder reducir ruido que corresponde a la forma de los objetos
- Calculamos el ángulo al que se encuentra el objeto

Todos esos datos los guardamos en una lista para poder tener toda la información de todas las detecciones que obtiene el modelo.

Segmentación

Como segundo paso vamos a calcular la máscara de la segmentación, con la cual podemos ver donde se encuentran las líneas de los carriles que queremos seguir. Para realizar esto vamos a realizar los siguientes pasos:

- Ejecutamos el modelo y guardamos mask, donde contiene todas las máscaras de los objetos que ha detectado

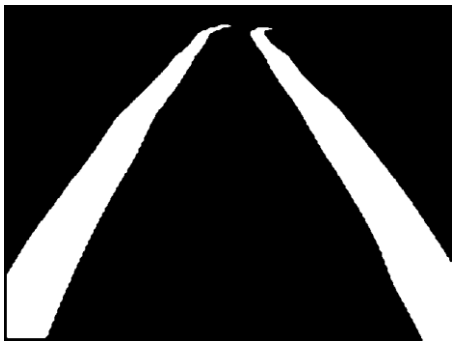
- Juntamos todas las máscaras para formar una única mascara y también la normalizamos para poder tener todos los valores entre 0 y 255
- Analizamos la máscara, que la función necesaria la comentaremos a continuación debido que la implementación es bastante compleja para

Después teniendo toda la información, vamos a responder al programa main, con toda la información que hemos calculado.

Antes de enviar esta información, para poder visualizar rápidamente las detecciones y las máscaras que hemos generado, vamos a publicar la imagen de la cámara con las cajas de las detecciones y también publicamos las máscaras. Esto lo hacemos debido que de esta manara podemos visualizar dichas imágenes en un navegador web y desde otro dispositivo.

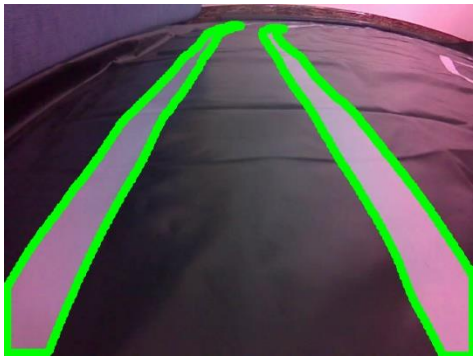
Función analizar mascaras

Primeramente, nos llega la máscara predicha por el modelo de segmentación, la máscara es una imagen en la cual se pueden ver las líneas predichas en color blanco y el fondo en color negro, tal y como se observa en la siguiente imagen:

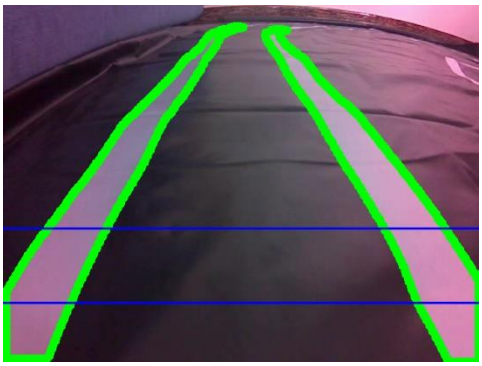


Ahora a partir de esta mascara vamos a calcular los puntos de interés que necesitamos para poder controlar el movimiento del robot para que pueda mantenerse en el carril. Los pasos que realizamos son los siguientes:

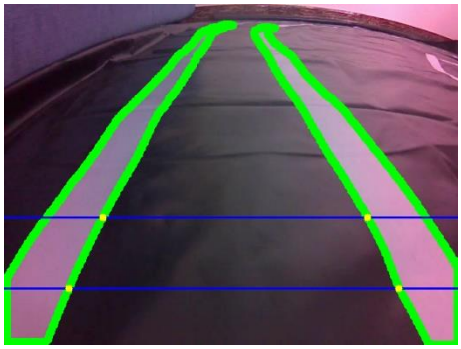
1. Calculamos los contornos de la imagen para poder tener las posiciones de las dos líneas blancas.



2. Creamos dos líneas imaginarias a diferentes alturas, con las que vamos a calcular los puntos del contorno que intersecan con dichas líneas.



3. Ahora calculamos los puntos que intersecan con dichas líneas. Como los contornos no son perfectos, para poder obtener un punto a la derecha y otro a la izquierda es necesario hacer un filtrado de esos puntos. Esto provoca errores que después se trasladaran a pasos siguientes.



4. Guardamos los puntos para poder realizar cálculos posteriores.

Servicio de conducción

Este código se encarga de obtener toda la información que le llega desde el código Main , la información que le llega es la información que dio como respuesta el servicio de analizar datos. El objetivo que tenemos con este código es calcular las velocidades necesarias para poder mover el robot. Para poder realizar esto el código realiza los siguientes cálculos:

Comprobación si hay elementos delante

Primeramente, comprobamos si hay algún elemento delante del robot, para poder discriminar si hay algún objeto delante, vamos a coger las distancias de las detecciones del modelo Yolo que usamos en la parte de analizar datos. Estos datos llegan al servicio en el mensaje que activa el servicio.

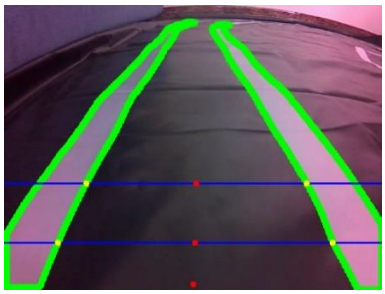
A partir de estas medidas comprobamos si se encuentran en el rango de peligro, si es asi, lo que realizamos es definir la velocidad lineal a 0 para que el robot se pare. Esto lo hacemos asi para no

complicar demasiado el problema al tener que hacer un comportamiento para poder esquivar los objetos y volver al carril.

Control para mantenernos dentro del carril

Ahora vamos a calcular la velocidad angular que necesitamos para mantenernos en el carril, solamente definimos la velocidad angular debido que la velocidad lineal la mantenemos constante. El cálculo de la velocidad angular es el siguiente

1. A partir de los puntos que hemos calculados anteriormente, vamos a calcular los centro entre dichos puntos, tenemos que tener en cuanta cuando uno de los puntos no existe, en eses casos vamos a definir el punto como el extremo de la imagen correspondiente a lado del punto. También definimos el centro de la imagen que lo tomamos como centro del robot.



2. Calculamos los ángulos entre el centro de la imagen y los diferentes centros.

$$\alpha = \arctan(\frac{CentroImagen - Centro}{AltoImagen - AlturaLineal})$$

3. Calculamos la velocidad angular para poder alinear el punto de cada una de las líneas con el centro de la imagen.

$$Va_i = K_p * \alpha$$

4. Calculamos la media de ambas velocidades para poder calcular la velocidad angular global.

$$Va = \frac{\sum Va_i}{2}$$

Resultados

En esto apartado vamos a comentar los resultados que hemos sacado después de la realización del desarrollo y las distintas pruebas que hemos hecho para poder obtener un resultado final.

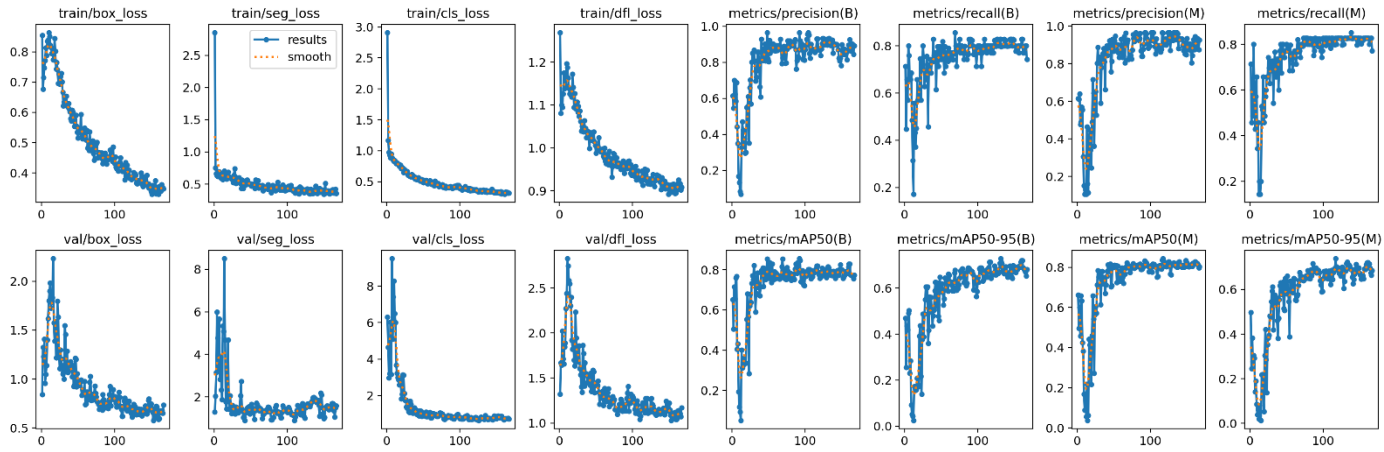
Entrenamiento YOLOv8 para segmentación

Primeramente, vamos a comentar los resultados del entrenamiento del modelo Yolov8 que usamos para segmentar las líneas de las carreteras.

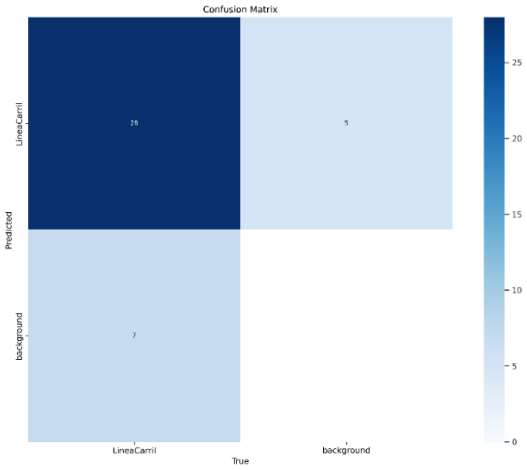
El dataset que hemos usado, son 76 imágenes, de las cuales después de aplicar varios filtros como flip o corte, hemos generado 182 imágenes de las cuales un 87% (159) para entrenar, 9%(16) para validar y 4%(7) para test.

En esta ocasión hemos entrenado con 300 épocas, pero el programa hizo un early stopping sobre las 170 épocas debido que el modelo empezaba a memorizar los datos. También comentar que el modelo que nosotros elegimos para poner en producción no es el modelo de la última epoca, sino que esta ocasión elegimos el modelo de la epoca con mejores resultados.

Las gráficas de las métricas de evaluación durante el entrenamiento es la siguiente:



Si ahora analizamos todas las gráficas podemos ver una tendencia en la cual el loss converge hacia cero, lo que quiere decir que el error disminuye, a pesar de que se puede apreciar que hay ruido, lo que empeora las medias. Ahora, si nos fijamos en las gráficas del mAP vemos que convergen cara 1, lo que nos dice que el algoritmo es capaz de detectar y clasificar correctamente todos los elementos, como a continuación vamos a ver en la matriz de confusión.



Como podemos ver, para las imágenes de test el modelo a clasificado correctamente 35 objetos, de los cuales 28 son línea de carril y 5 que son fondo. Vemos que el modelo se ha equivocado a la hora de clasificar tanto el fondo como la linea de carril, debido que ha clasificado 5 que son fondo en linea y 7 que son linea lo clasifico como fondo. Esto en nuestro caso no nos preocupa mucho debido que nuestro objetivo es la segmentación no la clasificación.

A continuación, vamos a visualizar las detecciones que ha hecho el modelo en las imágenes que tenemos para la validación:

Como podemos ver en los diferentes casos, hay una de las imágenes en la cual no ha detectado el carril o a detectado más de una vez la misma línea. Pero en general podemos ver que el resultado es bueno, por lo cual cuando lo pongamos en producción el modelo tendría que funcionar correctamente.

Resultados del comportamiento

Ahora vamos a comentar los resultados que hemos obtenido durante las pruebas que hemos realizado para poder comprobar la eficiencia del comportamiento que hemos desarrollado.

Problema de segmentación

Uno de los problemas que hemos encontrado, que ya se ha podido ver en el entrenamiento de la red de segmentación, es que en algunas ocasiones el modelo no segmenta las líneas de los carriles o en caso contrario segmenta más elementos.

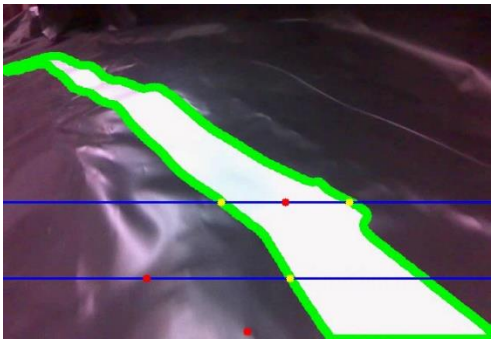
Como podemos ver en esta ocasión que no ha segmentado uno de los carriles, este es el peor caso debido que en esta ocasión, tal y como tenemos desarrollado nuestro algoritmo, no se encontraría el centro correcto con lo que añadiría ruido.

Si es el caso contrario cuando nos segmenta más elementos, debido a la forma en la que tenemos desarrollado nuestro proyecto, reducimos bastante el error, por lo cual afecta en el resultado final, pero es mínimo.

Elección de los puntos de los carriles

Uno de los problemas que tenemos es a la hora de seleccionar los puntos que corresponde a los diferentes carriles, debido a que hemos desarrollado un sistema básico el cual solamente busca el punto más cercano a la derecha y a la izquierda del centro de la imagen.

Debido a esto, un problema que tenemos es el siguiente:



Como podemos ver, cuando un contorno se encuentra cerca del centro, los puntos seleccionados para la derecha y para la izquierda pertenecen al mismo contorno.

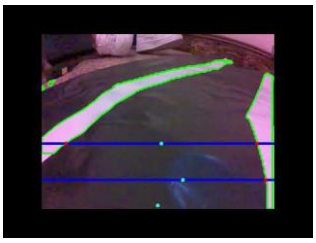
Resultados finales

Ahora vamos a analizar una de las pruebas que hemos ejecutado. A continuación, vamos a poder ver la trayectoria, las detecciones que ha realizado y las máscaras que se han creado con los cálculos que se han realizado. Los videos donde se pueden ver son los siguientes:

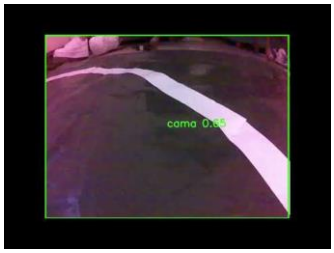
Trayectoria

Imagen

Mascaras



Detecciones



Ahora, si analizamos los diferentes videos, primeramente vemos que el robot intenta calcular la trayectoria correcta, pero sí que podemos ver que no es la perfecta, debido a que para eso habría que ajustar más el controlador proporcional o cambiar la forma de calcular la velocidad necesaria. Esto también puede venir condicionado debido a que en algunas ocasiones los puntos de las máscaras no son los correctos. Como podemos ver al final del video nos está ocurriendo uno de los errores que comentábamos antes.

En esta ocasión, las detecciones están fallando debido a que hace falsas detecciones, pero no influyen mucho ya que, cuando hace la correlación con las distancias, como no hay ningún objeto cercano a él no se cumple la condición de las distancias por consecuencia no se para el robot.

Conclusiones

En este último apartado vamos a comentar las conclusiones que hemos podido sacar después del desarrollo y análisis de los resultados que hemos obtenido.

El objetivo que estábamos buscando lo hemos realizado con éxito, a pesar de que sí que hemos encontrado algún error que no hemos arreglado debido que habría que profundizar en nuevas técnicas, como para el caso de seleccionar los puntos laterales de los carriles.

La elección de los modelos Yolo para hacer la detección y la segmentación ha sido una buena idea debido que hemos tenido buenos resultados, sobre todo a la hora de segmentar ya que con el entrenamiento que hemos hecho se obtienen muy buenos resultado. Para el caso de segmentación también hemos probado a usar el modelo Mask-RCNN, pero el problema que hemos encontrado es que cuando lo fuimos a implementar en el robot, había incompatibilidades en las versiones que nos impedía utilizarlo.

El uso de modelos basados en redes neuronales nos ha ayudado bastante para poder obtener el resultado que queríamos, debido que el uso de estos modelos es mucho más eficiente que el uso de visión clásica. Por lo cual, con estos modelos podemos hacer problemas más complejos de una forma más eficiente.

En esta ocasión, la mayoría de los desafíos que hemos encontrado los hemos resuelto, como el cálculo de la velocidad angular necesaria para mantenernos en el carril o la correlación entre las detecciones en la imagen y las medidas de los laser. Si que hay algunos desafíos que no hemos solucionado correctamente, pero hemos reducido la posibilidad de que aparezcan.

Si comentamos los resultados que hemos obtenidos, son buenos. El entrenamiento del modelo de segmentación fue el necesario para poder obtener resultados óptimos. Después, vemos que el comportamiento es el que buscábamos, aunque haya algún error que puede agregar ruido a la trayectoria, pero en general el resultado es el óptimo.

Las posibles mejoras que podemos hacer a este desarrollo es una mejora en los sensores, con lo que conseguiríamos una mejor precisión a la hora de obtener los datos. Una mejor optimización de los algoritmos, como el algoritmo de segmentación, debido que podemos hacer algún entrenamiento mejor en el cual añadiremos más imágenes en distintas situaciones para que le modelo pueda generalizar mejor. Implementación de otras tecnologías, sobre todo para la elección de los puntos y para la parte de conducción, en la que podríamos aplicar algoritmos de aprendizaje automático para poder mejorar la trayectoria.