

Conducción autónoma sobre plataforma real y simulada con seguimiento de carril e identificación de señales de tráfico y peatones mediante redes neuronales

Álvaro Mariscal Ávila

a.mariscal.2018@alumnos.urjc.es



Trabajo fin de grado

5 de julio de 2022

1. Mi nombre es ...
2. Voy a presentar mi trabajo fin de grado titulado: Conducción autónoma sobre plataforma real y simulada con seguimiento de carril e identificación de señales de tráfico y peatones mediante redes neuronales.



(CC) Álvaro Mariscal Ávila

*Este trabajo se entrega bajo licencia CC BY-NC-SA.
Usted es libre de (a) compartir: copiar y redistribuir el material en
cualquier medio o formato; y (b) adaptar: remezclar, transformar
y crear a partir del material. El licenciador no puede revocar estas
libertades mientras cumpla con los términos de la licencia.*

1. La presentación esta dividida en cinco partes.

1 Introducción

2 Objetivos

3 Plataforma de desarrollo

4 Sistema de conducción autónoma

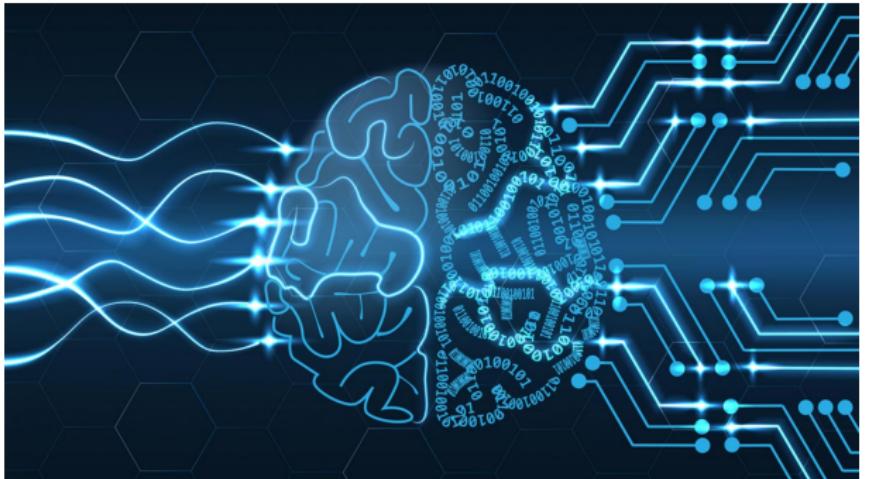
5 Conclusiones

1. Para empezar tenemos la introducción para hablar acerca del contexto general.

Introducción

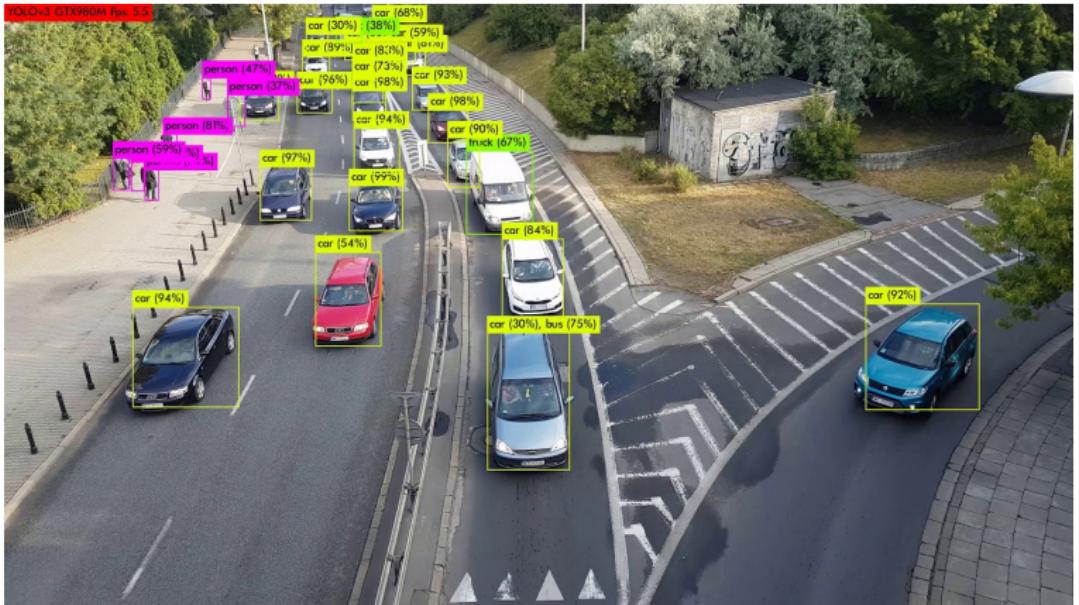
Inteligencia artificial

- Entender y emular el comportamiento humano.
- Dotar a sistemas de cierta inteligencia y de la capacidad de aprender.
- Visión artificial, aprendizaje automático o aprendizaje profundo.



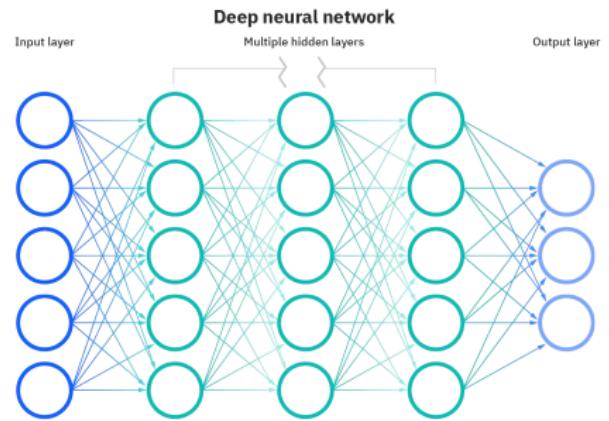
1. Inteligencia artificial: es la disciplina que trata de entender y emular el comportamiento humano.
2. Permite dotar a sistemas, entre los que se encuentran los robots, de cierta inteligencia y de la capacidad de aprender.
3. Multitud de ramas que parten de la IA como visión artificial, aprendizaje automático o aprendizaje profundo.

Visión artificial



1. Visión artificial utiliza sensores como las cámaras permiten obtener una información muy completa del entorno.
2. Tiene multitud de aplicaciones entre las que se encuentra la detección de objetos tal y como se observa en la imagen.
3. Para ello es necesario técnicas como Deep Learning o aprendizaje profundo.

Deep Learning



- Neocognitrón (1979): NN 5/6 capas **caracteres japoneses**.

1. Deep Learning se basa en el uso de redes neuronales artificiales.
2. Estas están inspiradas en el cerebro humano.
3. En el caso de la abstracción al mundo de la computación, una red está formada por una capa de entrada, una de salida y varias internas u ocultas.
4. Estas redes se entranan previamente mediante lo conocido como datasets. Que son grandes volúmenes de datos, ya sean imágenes, videos, sonidos etc.
5. Neocognitrón se considera el inicio del Deep Learning en 1979, que es una red neuronal con 5 o 6 capas para reconocer caracteres japoneses.
6. Además, decir que no existe límite claro para definir número de capas que debe tener una red neuronal para considerarla Deep Learning.

Vehículos autónomos



Figura: Camión autónomo nivel 5 de la marca Volvo.

- ① Asistencia a la conducción.
- ② Automatización parcial.
- ③ Automatización condicionada.
- ④ Automatización elevada.
- ⑤ Automatización completa.

1. En cuanto al contexto específico en el que se enmarca el trabajo, encontramos los vehículos autónomos que permiten circular sin conductor.
2. Si bien es cierto que aún no vemos este tipo de vehículos circulando por las ciudades, mucha de la tecnología necesaria ya está disponible en los vehículos actuales.
3. Existen diferentes niveles de autonomía. En el caso de la fotografía es un camión autónomo de la marca Volvo con nivel de autonomía 5, es decir, automatización completa en cualquier situación. Se han desarrollado pruebas en el puerto de Gotemburgo.
4. Y además, decir que la posibilidad de automatizar transportes de mercancías, sería un gran avance por las grandes distancias que recorren, incluyendo en algunos casos transporte internacional durante grandes intervalos de tiempo.

Autonomous Mobile Robots, AMRs



- Sucesores de los AGVs.

1. Son aquellos capaces de navegar por entornos dinámicos, conviviendo con humanos y sabiendo sobreponerse a situaciones para las que no habían sido programados explícitamente.
2. Son los sucesores de los AGVs. Estos requieren una cierta infraestructura dependiendo del tipo de guiado, ya sea filo-guiados, a través de pintura o a través de cualquier otra técnica que haga que ese vehículo solo pueda funcionar cuando se conoce la infraestructura previa que estará presente en el entorno de trabajo.
3. Kiva Systems: comprada por Amazon en 2012. Permiten automatizar sus almacenes en tareas de logística a nivel interno, maximizando la productividad y el almacenamiento, tanto en profundidad como en altura, y obviamente minimizando el coste en personal..

Objetivos

Descripción del problema

- Desarrollar un **coche autónomo** capaz de circular por un **circuito** en un entorno dinámico, interactuando con objetos propios de una ciudad en dos entornos distintos:
 - Entorno **simulado** con *Gazebo*.
 - Entorno **real** usando un robot con *Jetson Nano*.
- En ambos entornos se han de cumplir dos subobjetivos:
 - Seguimiento de **carril**.
 - Detección de **objetos**.

1. En cuanto a la descripción del problema.
2. Desarrollar coche autónomo capaz de circular por un circuito en un entorno dinámico, interactuando con objetos propios de una ciudad en dos entornos distintos:
3. Entorno simulado con Gazebo.
4. Entorno real usando un robot con Jetson Nano
5. En ambos entornos se han de cumplir dos subobjetivos:
6. Seguimiento de carril
7. Detección de objetos

Requisitos

- *GNU/Linux: Ubuntu 18.04 LTS.*
- Entorno simulado tarjeta gráfica dedicada: *NVIDIA* y *CUDA*.
- Entorno real *NVIDIA Jetson Nano*, económica con *GPU*.
- Lenguaje de programación *Python*.

1. En cuanto a los requisitos.
2. El sistema operativo utilizado será GNU/Linux, concretamente la distribución Ubuntu 18.04 LTS.
3. El entorno simulado requerirá la presencia de una tarjeta gráfica dedicada: NVIDIA y CUDA.
4. El entorno real requerirá un robot con la placa de desarrollo NVIDIA Jetson Nano, ya que esta es una de las placas con GPU más económicas.
5. El lenguaje de programación utilizado será Python.

Plataforma de desarrollo

NVIDIA Jetson Nano

- Placa de desarrollo de **bajo coste** con **GPU** dedicada.
- Arquitectura **Aarch64**, soporte para **GNU/Linux** y puertos **GPIO**.



1. La placa de desarrollo utilizada es NVIDIA Jetson Nano por su bajo coste y por tener una GPU dedicada.
2. Arquitectura Aarch64, soporte para GNU/Linux y puertos GPIO.

Componentes



1. En cuanto a los componentes utilizados para ensamblar el robot real.
2. Motores de bajo coste sin odometría.
3. Controlador de motores. Limitaciones motores a pares.
4. Batería 10500mAh para alimentar Jetson Nano.
5. Batería 7.4V para alimentar motores.
6. Cámara USB como sensor principal.
7. Adaptador Wi-Fi para conexión mediante SSH con el robot.
8. Chasis muy usado en proyectos relacionados con Arduino.

Software

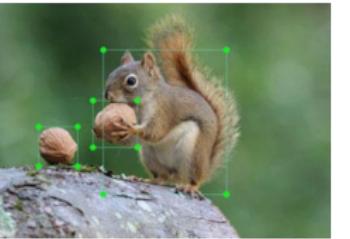


ROS

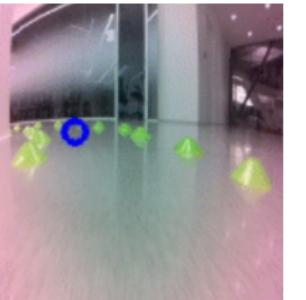
Qt + Python

1. Software utilizado.
2. Python: Lenguaje de programación interpretado que permite una programación rápida y tiene multitud de librerías compatibles.
3. FreeCAD para diseño de piezas en 2D y 3D.
4. Blender permite ensamblar robots y tiene plugins como Phobos que permiten definir su jerarquía.
5. Gazebo como simulador.
6. ROS como middleware para implementar el software utilizando la abstracción que proporciona.
7. PyQt para desarrollar interfaces de usuario.

Software relacionado con visión



1. Software relacionado con visión.
 2. OpenCV: librería de visión artificial con multitud de aplicaciones.
 3. YOLO: es un algoritmo de detección de objetos muy popular.
 4. LabelIMG: permite etiquetar imágenes componer un dataset y entrenar con él.
 5. Darknet: framework que permite entrenar y ejecutar redes neuronales.
 6. JetRacer: librería que permite entrenar y ejecutar una red para realizar seguimiento de carril.



1. Una vez descritos los objetivos y las plataformas utilizadas veamos el desarrollo en sí del proyecto.

Sistema de conducción autónoma

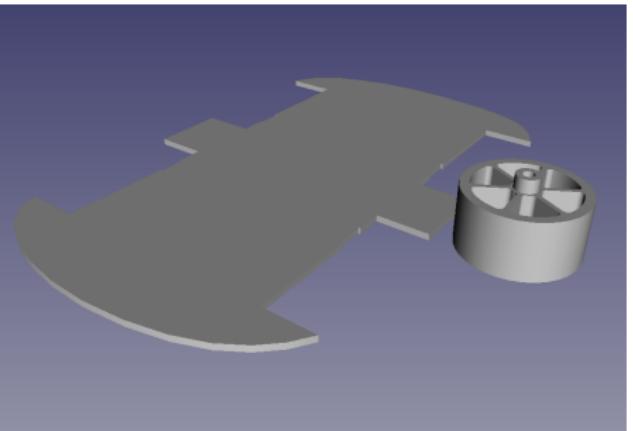
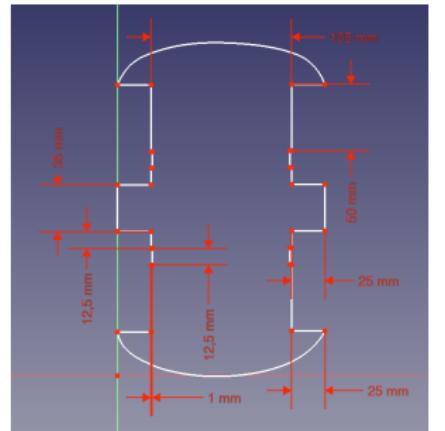
Modelo de la ciudad en Gazebo



- Reproducible en el entorno real.
- Semáforo y peatón dinámico.

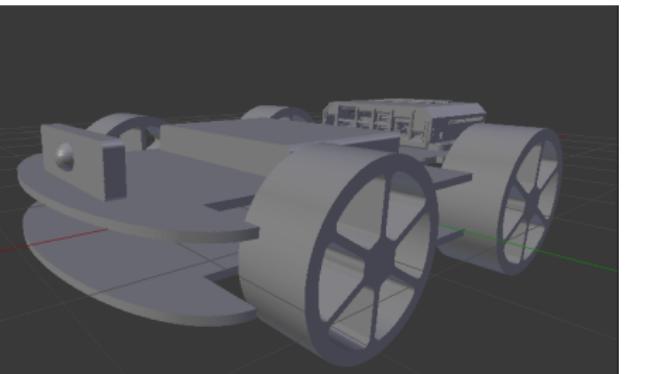
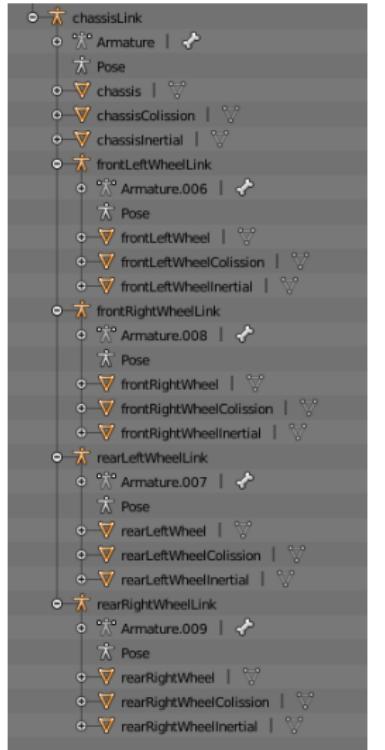
1. Modelo de la ciudad.
2. Uno de los requisitos es que sea reproducible en el entorno real, por eso partiendo de una ciudad de grandes dimensiones se reduce hasta conseguir lo que muestra la imagen.
3. La ciudad dispone de elementos dinámicos como un semáforo y un peatón y elementos estáticos como una señal de stop

Modelo del vehículo en *FreeCAD*



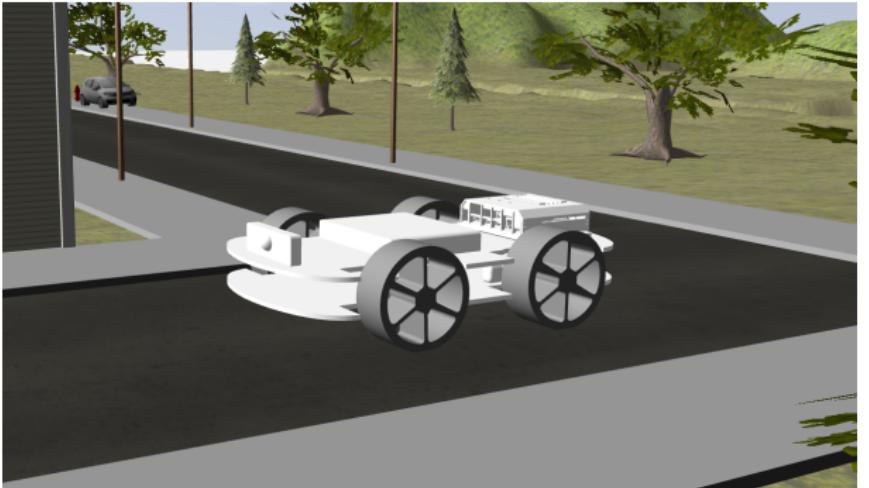
1. Modelo del coche autónomo diseñado en FreeCAD utilizando la herramienta Sketcher para realizar el diseño con cotas y restricciones de verticalidad y horizontalidad.
2. A continuación, se proporciona volumen a las piezas.

Modelo y jerarquía del vehículo en *Blender*



1. Una vez diseñadas las piezas en FreeCAD se exportan en STL para utilizarlas en Blender.
2. Se ensambla el robot y se compone su jerarquía con un link principal que es el cuerpo del robot y cuatro links dependientes de este que son las ruedas.
3. Se utilizan joints de tipo continuous para que las ruedas giren y así dotar de movimiento al robot.

Modelo del vehículo en Gazebo

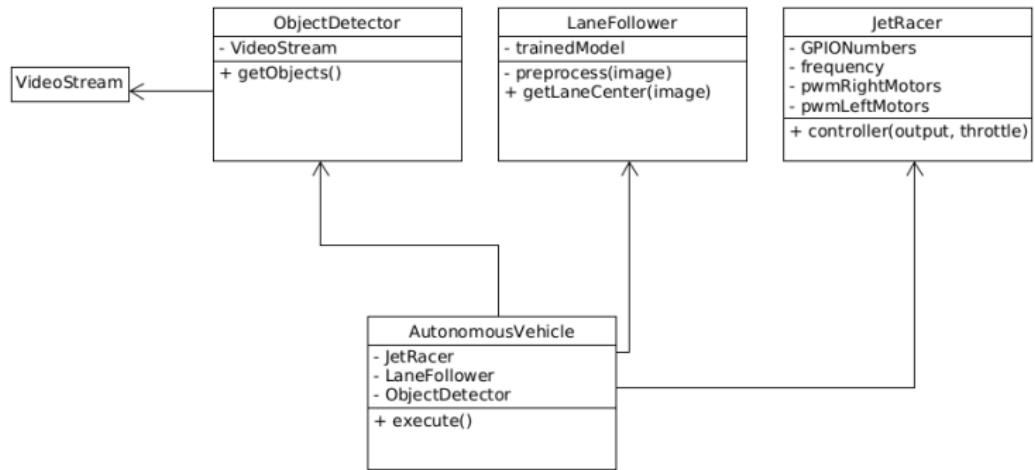


- *Plugins* para movimiento y cámara conectados con *ROS*.

1. Teniendo el modelo de la ciudad y el modelo del robot se ejecuta en Gazebo.
2. El robot se puede mover a través de topics de ROS y recibe la imagen que ve el robot a través de un subscriptor.

Diagrama de clases

1. Diagrama de clases.



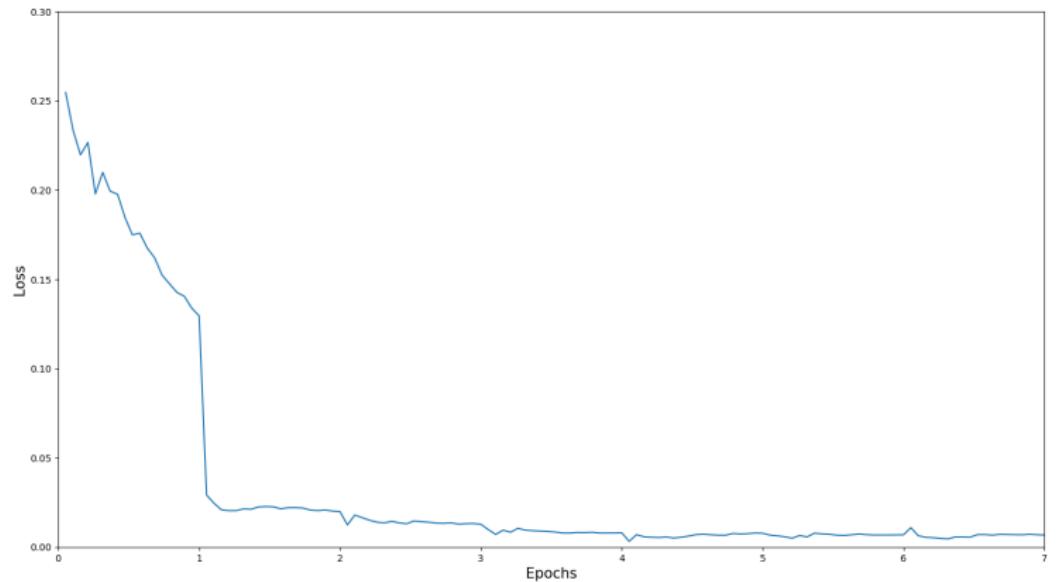
Seguimiento de carril



- Red *ResNet-18*: convolucional de 18 capas con **bloques residuales**.
- *Notebooks* de *Jupyter*.
- Guardado de cada imagen: *x_y_identificador_unico.jpg*.
- Dataset de 200 imágenes incluyendo situaciones **difíciles**.

1. En cuanto al seguimiento de carril se utiliza un red ResNet-18 preentrenada que es una red convolucional de 18 capas con bloques residuales que permiten incrementar el número de capas sin perder rendimiento ni precisión.
2. Utiliza notebooks de Jupyter.
3. Para entrenar, se guarda cada imagen haciendo click y se guarda con el siguiente nombre: *x_y_identificador_unico.jpg*.
4. Se utiliza un dataset de 200 imágenes incluyendo situaciones difíciles.

Entrenamiento red seguimiento de carril



1. Entrenamiento red seguimiento de carril.
2. Batch: número de ejemplos que se introducen en la red para que entrene de cada vez. Mayor más precisión más tiempo.

- En cada *epoch* se calcula el error cuadrático medio (*loss*).
- Propagación hacia atrás de los errores desde la capa de salida hasta la primera capa.

Salida red seguimiento de carril

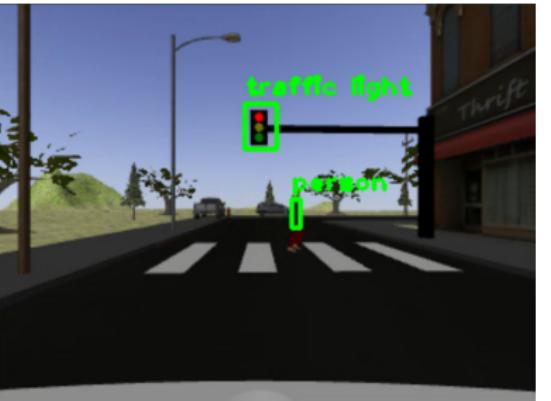
1. Entrenamiento red seguimiento de carril.



- 224 píxeles.
- Entorno experimental: autómata.

Detección de objetos

1. Detección de objetos.



- *YOLO V3 Tiny*.
- Objetos muy *genéricos*: alta probabilidad de detección.
- *Bounding Box*: clase y probabilidad.

Detección de semáforo

1. Detección de semáforo.

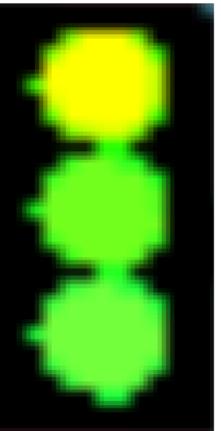
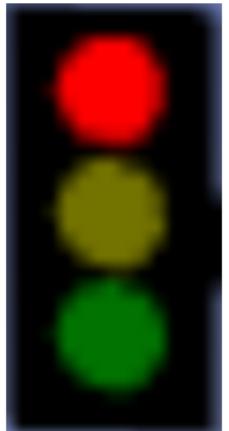
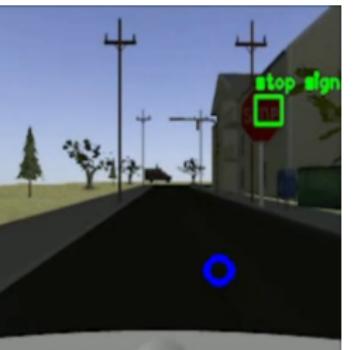
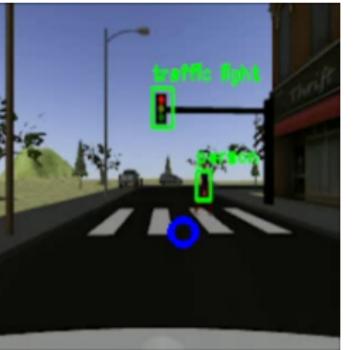


Figura: Imagen original Bounding Box, HSV y filtro de color.

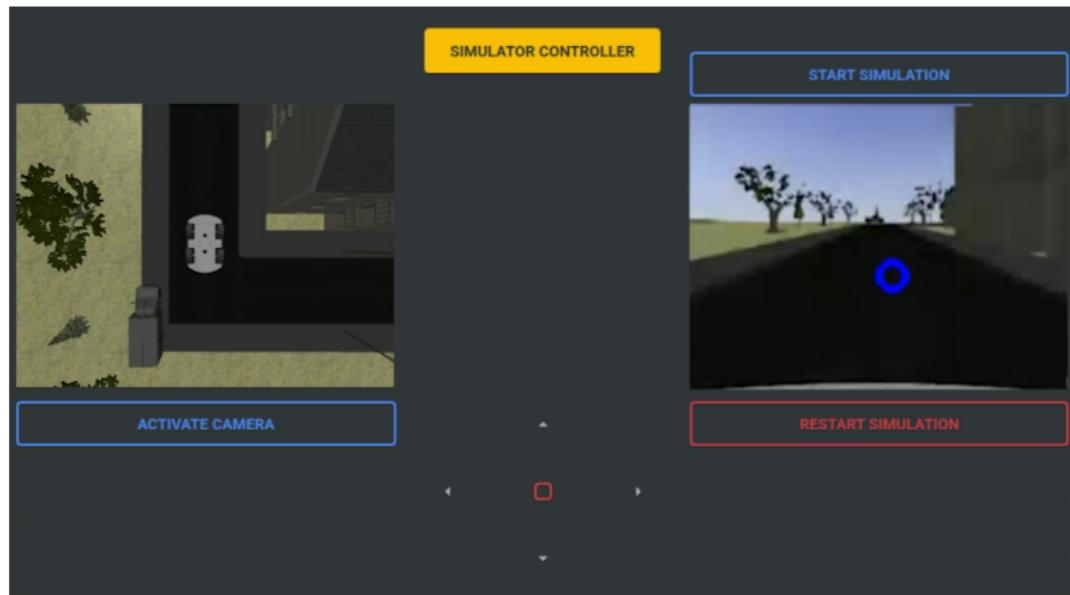
Ejecución en el simulador

1. Ejecución en el simulador.



Interfaz de usuario

1. Interfaz de usuario.



Entorno real

1. Entorno real.



Círculo inicial

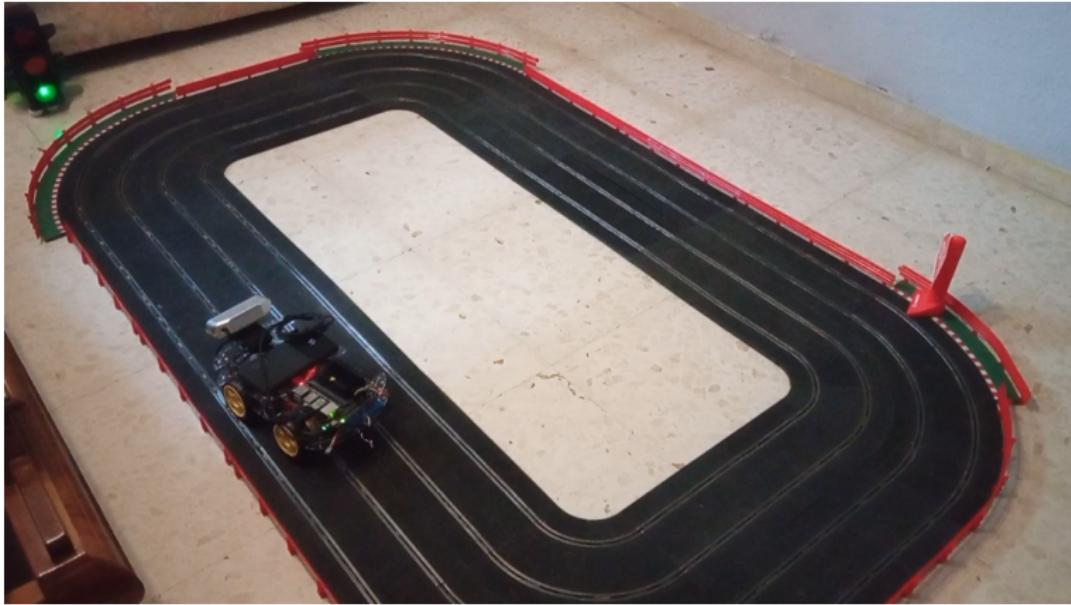
1. Circuito inicial.



- Entrenamiento con luz **artificial**.

Círculo con objetos

1. Circuito con objetos.
2. Alrededor de 2 metros de largo.



Dataset objetos reales

1. Dataset objetos reales.



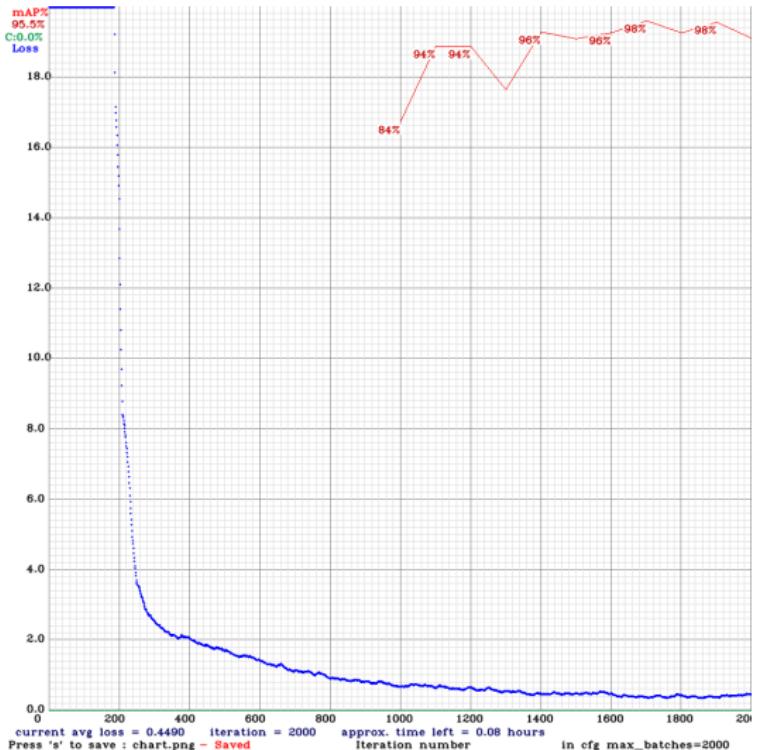
Entrenamiento red de detección de objetos

1. Entrenamiento red de detección de objetos.



- Portátil con *NVIDIA MX330* $\simeq 95^\circ C$.
- 2000 iteraciones $\simeq 3$ horas.
- Únicamente con *dataset* propio.

Entrenamiento red de detección de objetos



1. Entrenamiento red de detección de objetos.
2. Se muestra el *mAP* (*mean Average Precision*), valor que alcanza el 98 %. Esto es una métrica de precisión que se calcula cada 4 *epochs* (a partir de 1000 iteraciones) usando el *dataset* de validación. Una *epoch* es un número de iteraciones atendiendo a la siguiente fórmula: *número_imágenes_entrenamiento / batch*. En el caso de que este valor (expresado en porcentaje en la gráfica) se decrementara y el *loss* siguiese bajando, la red se estaría sobreajustando. Es lo que se conoce como *overfitting*.

Comparación probabilidad de detección

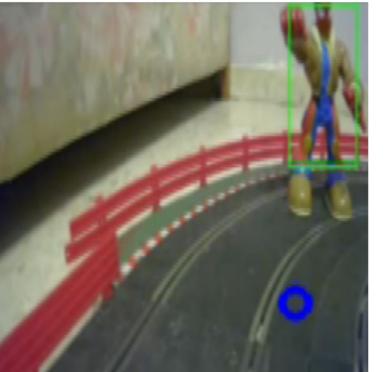
- Imagen **desconocida** para ambas redes.



- Comparación probabilidad de detección.
- Marca la diferencia entre detectar o no al peatón cuando el robot está en movimiento.

Ejecución en el entorno real

1. Ejecución en el entorno real.



Conclusiones

1. Para acabar esta presentación, vamos a repasar lo hecho, unas breves conclusiones y las líneas futuras.

Conclusiones

1. Conclusiones.

- Para lograr los objetivos se han utilizado dos **redes neuronales**:
 - Seguimiento de carril: *ResNet-18* combinada con un **controlador**.
 - Detección de objetos: *YOLO V3 Tiny* con **dataset** propio para aumentar la fiabilidad.
- Implementado en dos paquetes **ROS**, entorno simulado y real.
- Limitaciones: **ángulo** de la cámara y **resolución** de imagen en las *NN*.

Líneas futuras



1. Líneas futuras: mayor dataset, túnes, bosques, todo entorno donde exista algo similar a un camino a seguir. También sería interesante obtener los límites en izquierda y derecha del carril y no solo el punto central.

Conducción autónoma sobre plataforma real y simulada con seguimiento de carril e identificación de señales de tráfico y peatones mediante redes neuronales

Álvaro Mariscal Ávila

a.mariscal.2018@alumnos.urjc.es



Trabajo fin de grado

5 de julio de 2022

1. Vídeo.
2. Con las conclusiones doy por terminada mi presentación y quedo a su disposición.