

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE  
INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Detección de vehículos en videos de tráfico  
mediante aprendizaje profundo

GRADO EN INGENIERÍA DE  
TELEMÁTICA

ISMAEL AGUILERA CERVERA  
MÁLAGA, 2022



E.T.S. DE INGENIERÍA DE TELECOMUNICACIÓN, UNIVERSIDAD DE MÁLAGA

## **Detección de vehículos en videos de tráfico mediante aprendizaje profundo**

Autor: Ismael Aguilera Cervera

Tutor: Ezequiel López Rubio y Miguel Ángel Molina Cabello

Departamento: Lenguas y Ciencias de la Computación

Titulación: Grado en Ingeniería Telemática

Palabras clave: Inteligencia artificial, seguimiento, aprendizaje profundo, redes neuronales, detección, velocidad, anómalo.

### **Resumen**

La inteligencia artificial y el aprendizaje profundo dentro de la computación ha significado grandes mejoras en una gran cantidad de ámbitos industriales. Uno de los ámbitos en los que ha sido importante es en los sistemas de videovigilancia.

Este proyecto se centra en la estimación de la velocidad relativa a la cámara de objetos en movimiento. Para ello, este trabajo propone el uso de videos de tráfico para así poder utilizar las cámaras de videovigilancia que están en uso de las vías públicas y mediante la aplicación del aprendizaje profundo a esos videos, poder obtener unos resultados fiables.

Con motivo de esto se examinarán videos de tráfico tanto reales como simulados con diferentes comportamientos y analizando en cada una de las diferentes situaciones los resultados que se obtienen.

Después de lograr los resultados y analizarlos, se plantearán una serie de conclusiones y trabajos futuros que este trabajo de la posibilidad de realizar o proseguir.

### **English version of the title**

Author: Ismael Aguilera Cervera

Supervisor: Ezequiel López Rubio y Miguel Ángel Molina Cabello

Department: Lenguas y Ciencias de la Computación

Degree: Grado en Ingeniería Telemática

Keywords: artificial intelligence, tracking, Deep learning, neural networks, detection, speed, anomalous.

### **Abstract**

Artificial intelligence and deep learning in computing has meant great improvements in many industrial fields. One of the areas where it has been important is in video surveillance systems.

This project focuses on estimating the velocity relative to the camera of moving objects. To do this, this work proposes the use of traffic videos to use the video surveillance cameras that are in use on public roads and by applying deep learning to these videos, to obtain reliable results.

For this purpose, both real and simulated traffic videos with different behaviours will be examined and the results obtained in each of the different situations will be analysed.

After achieving the results and analysing them, a series of conclusions will be drawn and future work that this work will be able to carry out or continue.

# Agradecimientos

Me gustaría agradecer a mis tutores del Trabajo de Fin de Grado, Ezequiel y Miguel Ángel, por guiarme y enseñarme durante todo el trabajo, y por su compromiso y ayuda que me han dado a lo largo de este tiempo.

También a mis profesores, amigos y compañeros que he tenido a lo largo de estos 4 años que ha durado mi formación.

Y por último agradecer a mi familia, que siempre me han apoyado cuando más me ha costado avanzar.



*Quiero dedicar este trabajo a mis padres y hermano,  
En especial a mi abuelo Antonio, siempre te recordaré.  
Por su apoyo y confianza en mí.*





# Contenido

<b>Capítulo 1. Introducción .....</b>	<b>1</b>
1.1. Motivación .....	1
1.2. Objetivo.....	2
<b>Capítulo 2. Marco teórico.....</b>	<b>4</b>
2.1. Redes neuronales.....	4
2.1.1. Estructura de los modelos de las redes neuronales .....	4
2.2. Redes de aprendizaje profundo .....	5
2.3. Redes Neuronales Convolucionales.....	6
2.3.1. Capas convolucionales .....	7
2.4. Backpropagation.....	7
2.5. Detección de objetos.....	8
2.6. Algoritmo RANSAC .....	8
<b>Capítulo 3. Tecnologías utilizadas.....</b>	<b>9</b>
3.1. Lenguaje de programación.....	9
3.2. Entorno de trabajo .....	10
3.3. YOLO .....	11
3.4. CARLA Simulator.....	12
3.5. OBS Studio .....	13
<b>Capítulo 4. Desarrollo del código.....</b>	<b>15</b>
4.1. Detección de vehículos .....	15
4.2. Seguimiento de vehículos .....	17
4.3. Creación de videos.....	18
4.3.1. Elección de mapa.....	18
4.3.2. Generación de actores.....	19

4.3.3. Comportamiento de los vehículos.....	21
4.4. Estimación de la velocidad.....	23
<b>Capítulo 5. Análisis de los resultados.....</b>	<b>26</b>
5.1. Videos reales.....	26
5.2. Videos simulados .....	33
<b>Capítulo 6. Conclusiones y trabajo futuro.....</b>	<b>38</b>
6.1. Conclusiones .....	38
6.2. Trabajo futuro .....	39
<b>Apéndice A. Uso del sistema .....</b>	<b>41</b>
A.1. Introducción a Google Colab .....	41
A.1. Introducción a CARLA Simulator.....	42
<b>Capítulo 7. Referencias.....</b>	<b>43</b>



## Lista de Acrónimos

CARLA	Car Learning to Act
CPU	Unidad de Procesamiento Central
GPU	Unidades de Procesamiento de Gráficos
IA	Inteligencia Artificial
OBS	Open Broadcaster Software
RANSAC	Random Sample Consensus
YOLO	You Only Look Once

# Capítulo 1. Introducción

## 1.1. Motivación

Hoy en día cada vez se usa más una toma de decisiones basada en los datos debido al tiempo que vivimos en los que cada vez está más avanzada la tecnología. Al tener un gran volumen de datos e información que manejar, el ser humano no es capaz de llevar a cabo trabajos con esa cantidad de información, por lo que se dio la necesidad de emplear los procesadores de los ordenadores para realizar esas tareas.

La inteligencia artificial y el aprendizaje automático está causando un gran impacto en la revolución de las tecnologías, debido a esto los ordenadores pueden manipular grandes cantidades de datos. El aprendizaje profundo usa conjuntos de algoritmos para aprender sobre dichos datos y utiliza lo aprendido para mejorar la toma de decisiones respecto a ellos.

Estas características se pueden aplicar por ordenador mediante métodos a la videovigilancia y la seguridad, ya que son campos con una gran investigación. Las vías de tráfico cuentan cada vez más con cámaras de video en lugares públicos como pueden ser calles, autovías y carreteras para la vigilancia del tráfico. Sobre este contexto, vamos a aplicar dichos procedimientos para generar una detección automática de objetos.

En particular, en este trabajo se va a hablar sobre la detección de vehículos y el seguimiento de sus trayectorias. Existen varios sistemas que pueden usarse para la detección de vehículos, pero en este trabajo vamos a enfocarnos en la detección y seguimiento de vehículos en videos de tráfico y de ahí poder estimar la velocidad relativa a la cámara de objetos en movimiento.

Para realizar esto se necesitará una YOLO (“Solo se mira una vez”) que es un algoritmo pre entrenado de detección de objetos que divide las imágenes de un sistema de cuadrícula, en la que cada celda de dicha cuadrícula debe detectar objetos dentro de sí

misma. YOLO es uno de los algoritmos de detección de objetos más famosos debido a su precisión y a su velocidad.

Gracias a los avances de la neuro computación, se ha podido dar la detección de objetos en capas convolucionales que permiten dentro de una imagen detectar objetos identificando su posición y el tipo de objeto.

En base a esto, se puede detectar trayectorias anómalas y estimar velocidades de los vehículos en los videos de tráfico y se podría usar esta información para utilizarla en aplicaciones como por ejemplo sería la seguridad vial.

Una vez presentado este contexto dentro de la inteligencia artificial, se va a demostrar a lo largo del proyecto que la tecnología puede ir de la mano del tráfico vial sugiriendo una serie de trabajos futuros e implementaciones que pueden enfocarse en esta rama.

## 1.2. Objetivo

El objetivo de este trabajo es desarrollar un sistema de identificación de vehículos que se pueda utilizar en cualquier video de tráfico de una manera eficiente y en base a eso estimar la velocidad relativa a la cámara de objetos en movimiento. Para ello se debe apoyar en la YOLO, en concreto la versión cinco, que es una familia de arquitecturas y modelos de detección de objetos pre entrenados en un conjunto de datos, que permitirá obtener las coordenadas exactas, el tipo de vehículo que es, y la fiabilidad con la que detectará dicho vehículo. A parte de hacer la detección de los vehículos correspondiente se les deberá hacer una predicción de su trayectoria o seguimiento de la trayectoria usando un *tracker* de la biblioteca *norfair*.

También se realizará unos videos con el programa “CARLA simulator” (Car Learning to Act) para generar tráfico vial de vehículos normales, pero que también se muestren vehículos con trayectorias anómalas como vehículos adelantando, vehículos que aceleren, vehículos que frenen o vehículos circulando a gran velocidad, todo esto en una autovía con lo que se deberá buscar un mapa correspondiente y unas condiciones adecuadas para la realización del video y posteriormente usarlo para detectar los vehículos y predecir su trayectoria en diferentes contextos y escenarios.

Finalmente se podrá obtener esa velocidad de la que se habló antes mediante los parámetros que se obtengan del seguimiento de los vehículos y mediante una regresión lineal y unos cálculos matemáticos para lograr calcular la pendiente de esa recta de regresión se logrará conseguir la velocidad estimada para cada identificador que sea detectado.

Una vez obtenida la velocidad estimada se analizarán los resultados obtenidos y se llegará a una conclusión para cerrar el proyecto y se propondrán unos posibles trabajos futuros que puedan seguir utilizando la base del proyecto realizado.

## Capítulo 2. Marco teórico

### 2.1. Redes neuronales

Las redes neuronales [1] son un sistema de computación que se basa en las redes neuronales biológicas de los seres vivos.

Son capaces de encontrar relaciones para crear patrones de forma inductiva por medio de algoritmos de aprendizaje que se basan en datos existentes.

Las redes neuronales artificiales se basan en la relación que existe entre el comportamiento y función del cerebro humano (sistema nervioso), ya que está compuesto por redes de neuronas biológicas que al realizarse una conectividad entre ellas tiene una gran capacidad de procesamiento.

Mediante un algoritmo de aprendizaje las redes neuronales artificiales realizan un ajuste de parámetros y arquitectura para poder minimizar la función de error.

Según el algoritmo de aprendizaje puede ser:

- **Aprendizaje supervisado:** Los datos de entrada están etiquetados, se conoce el resultado correcto para cada dato de entrada, con ello el algoritmo aprende y genera un modelo capaz de predecir el valor de salida
- **Aprendizaje no supervisado:** Los datos de entrada no están etiquetados, el algoritmo intenta realizar de alguna forma la relación de entrada.

#### 2.1.1. Estructura de los modelos de las redes neuronales

El elemento principal de una red neuronal [2] es la neurona, cuyo modelo podemos ver en la Figura 1. La suma de las  $n$  entradas  $x_j$  de la neurona, que tienen los pesos sinápticos  $w_{ij}$ , da lugar al potencial postsináptico de la neurona. Los pesos sinápticos  $w_{ij}$ , miden la intensidad de la interacción entre dos neuronas



que están conectadas por el enlace. En el siguiente paso se introduce a la función de transferencia ( $f$ ), la diferencia entre el potencial postsináptico y el umbral  $\theta_i$ , obteniéndose una salida  $y_i$ .

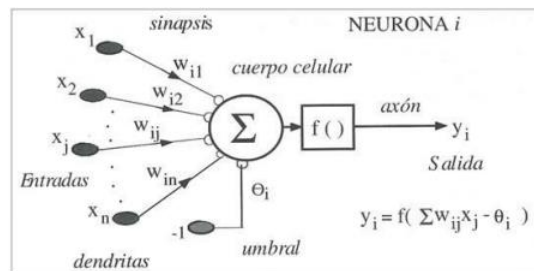


Figura 2.1: Modelo para neurona artificial. Fuente: [22]

### ***Tipos de arquitecturas***

- Red monocapa: Es la red neuronal más sencilla ya que solo se tiene una capa de neuronas que proyectan las entradas a una capa de neuronas de salida en la que se realizan los cálculos.
- Red multicapa: Es igual que la red monocapa con la diferencia de que existen un conjunto de capas intermedias entre la de entrada y la de salida.
- Red recurrente: La diferencia respecto a este tipo de red y las anteriores son lazos de realimentación en la red

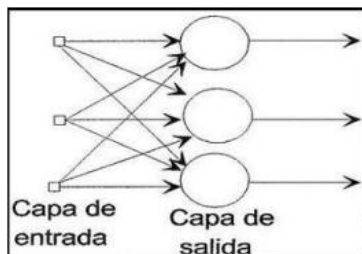


Figura 2.2a: Red monocapa.

Fuente: [2]

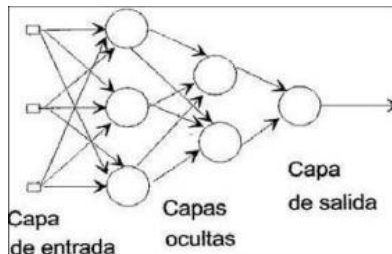


Figura 2.2b: Red multicapa.

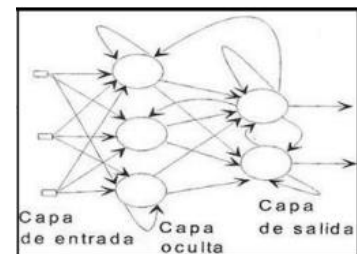


Figura 2.2c: Red recurrente.

## **2.2. Redes de aprendizaje profundo**

Las redes neuronales de aprendizaje profundo son un tipo de redes neuronales artificiales, su principal característica es que poseen bastantes capas ocultas intermedias. Su intención es intentar parecerse al funcionamiento del cerebro humano, interconectando las neuronas de una capa con las de la siguiente capa,

por lo que se realizaría un aprendizaje en la que en cada etapa se modifica los pesos de las neuronas de cada capa.

Para este aprendizaje es necesario una GPU [3] (Unidad de procesamiento gráfico) debido a que necesitamos hacer muchas operaciones al mismo tiempo en lugar de hacerlo una después de otra como haría una CPU (Unidad de Procesamiento Central).

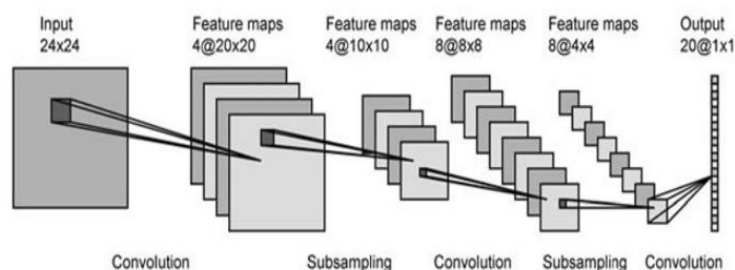
En resumen, esta estrategia se caracteriza por dar un análisis crítico a nuestras ideas, para favorecer la comprensión para que más tarde puedan ser utilizadas en solución de problemas en distintos contextos.

Principalmente el aprendizaje profundo se usa para la detección de objetos y la clasificación de ellos.

## 2.3. Redes Neuronales Convolucionales

Las redes neuronales convolucionales se usan principalmente para procesar imágenes [4], aprenden relaciones de entrada-salida donde la entrada es una imagen.

Las redes neuronales convolucionales [5] se basan en múltiples capas de filtros convolucionales de una o más dimensiones. Estas redes aprenden a reconocer objetos dentro de imágenes mediante un entrenamiento previo en el que añadimos una gran cantidad de muestras con las que la red pueda captar una relación de cada objeto para identificarlo.



**Figura 2.3: Proceso red convolucional. Fuente: [5]**

La convolución no es más que aplicar sobre la matriz de entrada de la capa filtros para obtener nuevas matrices que representan los patrones detectados.

La convolución consiste en multiplicar escalarmente el kernel por una región del mismo tamaño que la matriz de entrada [7], obteniéndose el valor de la matriz de salida en esa posición, este proceso se repite desplazando el kernel hasta

rellenar todas las posiciones de la matriz de salida. Para simplificar la convolución, la región seleccionada de la matriz de entrada se le aplica un Zero Padding, que es una técnica que consiste en rodear la matriz de ceros. El proceso explicado esta representado en la Figura 4.

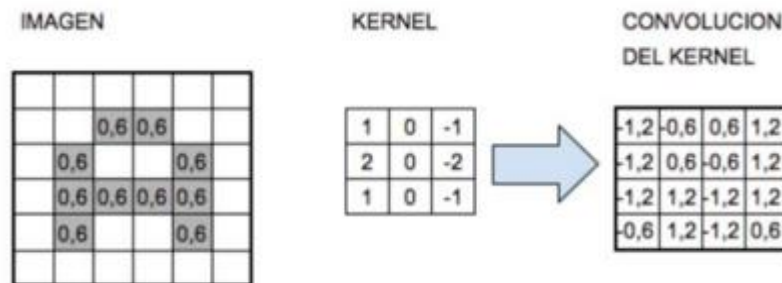


Figura 2.4: Proceso de filtrado. Fuente: [7]

### 2.3.1. Capas convolucionales

Como se ha dicho previamente, las redes neuronales convolucionales contienen varias capas ocultas especializadas y con una jerarquía distinta según la capa.

Las primeras capas pueden detectar líneas, curvas y se especializan para llegar a capas más profundas, las cuales, reconocen formas más complejas.

## 2.4. Backpropagation

Las redes neuronales usan un proceso de aprendizaje que se basa en realizar modificaciones poco significativas en los pesos de entrada de las neuronas hasta dar con la combinación que devuelva el resultado esperado [11]. Debido a la baja capacidad de procesamiento de las computadoras a mitad del siglo XX, las redes neuronales cayeron un poco en el olvido.

A finales del siglo XX se presentó el algoritmo de retro propagación (*backpropagation*) que fue muy popular e importante dentro del aprendizaje automático [12].

Principalmente lo que destacaba de este algoritmo por qué pese a tener una tecnología muy sencilla, su rendimiento era muy potente. Su función era definir una función de pérdidas para que comparara la salida que se obtenía con el resultado esperado, y mediante el uso de la técnica de descenso por gradiente

(que es un algoritmo que estima numéricamente el lugar donde la función genera sus valores más bajos [13]), se obtiene un conjunto de pesos que dan lugar a un resultado óptimo de la operación.

## 2.5. Detección de objetos

La detección de objetos es un campo importante en la inteligencia artificial [6], es usada para detectar e identificar distintos objetos (personas, automóviles, objetos...) en imágenes o videos y determinar su posición.

La principal técnica usada para obtener los datos es la segmentación de la imagen en rectángulos y en base a eso calcular la probabilidad de que exista un objeto en ese rectángulo. Esta tecnología ya aplica a multitud de áreas, como son la conducción autónoma, la robótica o la video vigilancia [8].

El resultado que se suele obtener son las coordenadas en pixeles donde se encuentran los rectángulos que se mencionan en el párrafo anterior. Los algoritmos más usados que suelen utilizarse son Faster R-CNN y YOLO9000

## 2.6. Algoritmo RANSAC

Para estimar la velocidad relativa a la cámara de objetos en movimiento se ha calculado el diámetro angular que tiene una dependencia lineal con respecto al tiempo para ver como varía esa cantidad.

La pendiente de la curva que expresa esa cantidad es proporcional a la velocidad relativa a la cámara del objeto. En este proyecto se ha usado el algoritmo RANSAC (Random Sample Consensus) para estimar esa pendiente ya que existe esa dependencia lineal que hemos mencionado antes del diámetro angular que es el inverso de la raíz cuadrada del área de la *bounding box* de cada objeto identificado.

Se puede obtener la velocidad en metros por segundo multiplicando el diámetro del vehículo por los coeficientes de la regresión lineal que hemos generado de la recta de regresión creada usando los conceptos de los que se han hablado.

## Capítulo 3. Tecnologías utilizadas

Después de ser explicados los conceptos del trabajo que las redes neuronales realizan con las imágenes, se van a usar una serie de herramientas y tecnologías para el desarrollo del proyecto.

### 3.1. Lenguaje de programación

A la hora de iniciarse en el mundo de la inteligencia artificial una de las primeras y principales decisiones que hay que tomar es sobre el lenguaje de programación que se va a utilizar.

Debemos tener en cuenta que la programación orientada a la inteligencia artificial tiene aspectos muy distintos de la programación convencional, para el caso de uso de la inteligencia artificial el programador debe enseñar al procesador a programarse solo.

Entre los lenguajes de programación más usados para trabajar dentro de la inteligencia artificial destacan Python y R [9]



**Figura 3.1: Lenguaje de programación Python. Fuente: [20]**

Python a medida que avanza el tiempo se está convirtiendo en un lenguaje más usado y popular dada su sencillez y versatilidad. La inteligencia artificial es uno de sus principales usos ya que posee un gran ecosistema de bibliotecas y

frameworks ya hechos para el aprendizaje automático como por ejemplo PyTorch, Pandas, NumPy o TensorFlow, por esto se ha optado por utilizar Python en la versión 3.6 para realizar este proyecto.

También se usarán distintas bibliotecas que permite usar el lenguaje Python para facilitar el desarrollo de la programación, a continuación se va a explicar el funcionamiento de algunas de estas bibliotecas:

NumPy es una librería especializada en el cálculo numérico y el análisis de los datos [10], aporta vectores o arrays y herramientas matemáticas sobre ellos.

Pandas es una librería que se emplea para el análisis de estructuras de datos, se ha usado principalmente para la lectura de los archivos csv que se han utilizado en el proyecto. Permite la lectura de estos y el tratamiento de sus datos.

PyTorch ofrece soporte a la programación de redes neuronales en Python, tiene dos rasgos importantes que son por un lado los “tensores”, que es una estructura de datos multidimensional que permite almacenar una colección de números. Por otro lado ofrece el “backpropagation” explicado en el capítulo anterior.

OpenCV proporciona un marco de trabajo de alto nivel para el uso de aplicaciones de visión por ordenador en tiempo real como puede ser el análisis de imágenes.

Norfair es una biblioteca de Python ligera y personalizable para el seguimiento de objetos 2D en tiempo real. Permite agregar capacidades de seguimiento a cualquier detector con pocas líneas de código [16].

Para la parte de estimar la velocidad se utilizó también la librería Matplotlib para aplicar una serie de formulas y procedimientos para la obtención de los valores.

## 3.2. Entorno de trabajo

El entorno de trabajo que se ha utilizado para este proyecto es Google Colab. Colab es un entorno de cuaderno Jupyter<sup>1</sup> de uso gratuito que se ejecuta

---

<sup>1</sup> Interfaz de código abierto que permite la ejecución de código a través del navegador en varios lenguajes.

completamente en la nube [14]. Esta plataforma no requiere una configuración y los usuarios pueden editar simultáneamente cuadernos que se creen.

Google Colab es también compatible con muchas bibliotecas importantes de aprendizaje automático que permiten cargar fácilmente en el ordenador, además, tiene un servicio gratuito con GPU. Es por estas razones principalmente por lo que hemos usado esta plataforma como entorno de trabajo para abordar este proyecto.

En esta plataforma se ha llevado a cabo la creación del algoritmo que detecta los vehículos y hace un seguimiento de la trayectoria al pasarle un video al programa.

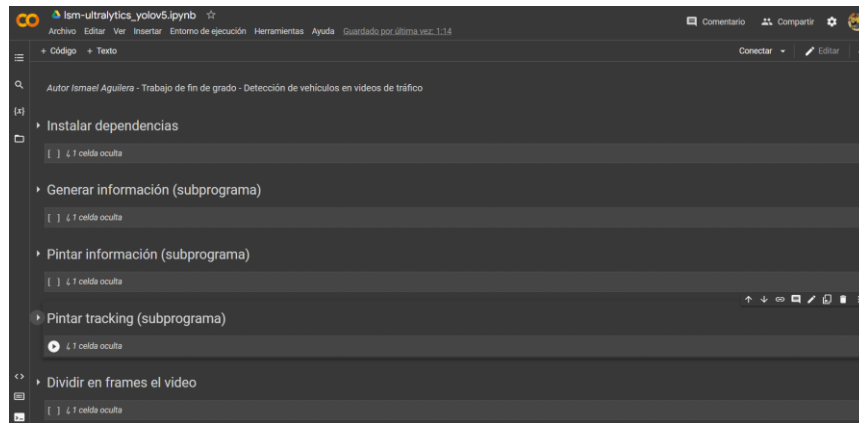


Figura 3.2: Entorno de trabajo de Google Colab.

### 3.3. YOLO

YOLO es un sistema del estado del arte que utiliza una red neuronal convolucional para la detección de objetos en tiempo real [15]. Se aplica la red neuronal a una imagen, la red divide la imagen en regiones y predice múltiples bounding box y la probabilidad de detección de las clases de entrenamiento. El sistema nos devuelve el identificador del objeto que detectó, su confianza y el tiempo de ejecución.

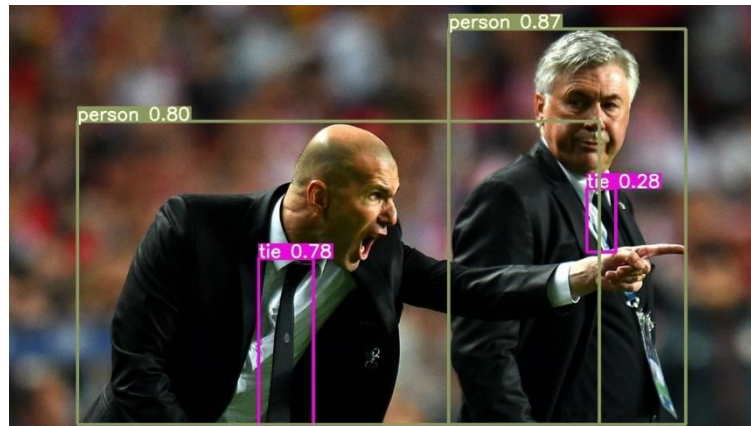


Figura 3.3: Detección de objetos con YOLO. Fuente: [19]

En este proyecto se va a utilizar la versión cinco de YOLO (YOLOv5) que está implementada en Pytorch que se ha explicado en el apartado anterior.

### 3.4. CARLA Simulator

Para poder comprobar todas las posibilidades y casos de la detección de objetos se han realizado una serie de videos mediante un simulador CARLA que es un simulador de conducción autónoma de código abierto (*open source*) que permite programar los tipos de vehículos, su comportamiento y el escenario en el que transitan los vehículos, es decir, simula una amplia gama de formas de conducción y repite situaciones de peligro de manera infinita para ayudar al aprendizaje.



Figura 3.4: Escenario de CARLA simulator. Fuente: [23]



CARLA fue creado desde cero para servir como una API modular y flexible para abordar una variedad de tareas involucradas en el problema de la conducción autónoma. Uno de sus principales objetivos es ayudar a democratizar la I+D en la conducción autónoma para que sirva como una herramienta de uso sencillo y personalizable por los usuarios.

El simulador de carla funciona mediante una arquitectura de cliente-servidor escalable [24]. El servidor es el responsable de todo lo relacionado con la simulación en sí como es los sensores, los actores o el cálculo de la física. Cuando se trata de aprendizaje automático la mejor opción siempre es ejecutar el servidor con una GPU dedicada. Por otro lado, el cliente controla la lógica de los actores en escena y establece las condiciones del mundo.

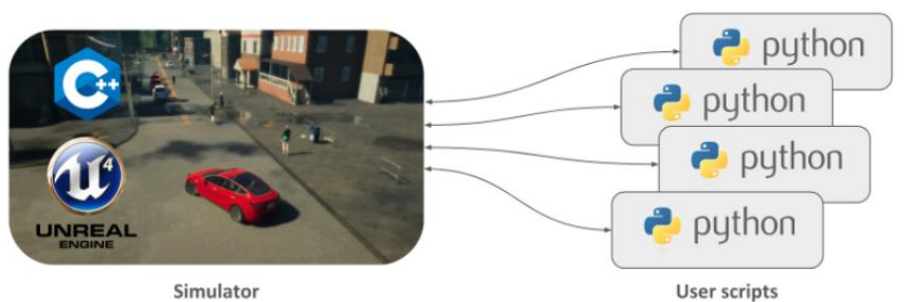


Figura 3.5: Arquitectura cliente-servidor. Fuente: [24]

### 3.5. OBS Studio

Para poder grabar las simulaciones que se han realizado con el programa CARLA simulator se ha optado por usar un programa externo de grabación de pantalla para facilitar el trabajo de grabación de las simulaciones, en el caso de este trabajo se ha elegido OBS studio en su versión 26.0.2.



Figura 3.6: Logo de OBS Studio. Fuente: [26]

OBS studio (*Open Broadcaster Software*) es un software libre y de código abierto para grabación de video [26]. Se trata de una aplicación con características profesionales y fácil de utilizar.

## Capítulo 4. Desarrollo del código

En este apartado se va a explicar la parte del desarrollo del código que se ha realizado para conseguir los algoritmos de detección y seguimiento de vehículos y su posterior estimación de la velocidad relativa a la cámara de objetos en movimiento. Para ello se han utilizado las distintas herramientas explicadas en el apartado anterior pero ahora vamos a profundizar más en cómo se ha usado dichas herramientas y la finalidad con la que se han empleado.

### 4.1. Detección de vehículos

Para la detección de vehículos, cómo se ha dicho en el apartado anterior, se ha utilizado el entorno de trabajo Google Colab y el lenguaje de programación Python.

En primer lugar, se han instalado todas las dependencias necesarias para empezar a programar, como son la YOLOv5 y montar el drive que se va a utilizar como almacenamiento de la información que se utilizará y posteriormente que se guardará.

Después se va a proceder a la división del video entrante a fotogramas (*frames*), el primer paso es usar la librería cv2 del paquete OpenCV (explicado en el capítulo anterior) para obtener el video y la cantidad de fotogramas que tiene el mismo video.

Una vez obtenidos se va a aplicar un comando de 'FFmpeg' que es una colección de software libre para grabar, convertir y editar audio y video que usa una línea de comandos [16].

Este comando nos permite sacar uno a uno los fotogramas del video y guardarlos en una carpeta para su posterior análisis y uso en más algoritmos.

```
command = str('ffmpeg -i ' + '"' + str(fname) + '"' + ' ' + '-q:v 1' + ' ' + '-start_number 0' + ' ' + '"' + str(saveDirFileNames) + '"')
```

El siguiente paso que se necesita realizar es la generación de información de cada fotograma, es decir, pasar la Yolov5 por cada fotograma que guardamos en el paso anterior para así conseguir detectar la información en todos los fotogramas. Simplemente le pasaremos cada imagen con la Yolov5 pre entrenada y nos devolverá las coordenadas para dibujar la caja delimitadora o *'bounding box'*, la confianza de la detección y la clase del objeto detectado. Toda esta información se guardará en un archivo de extensión *'csv'* para cada uno de los fotogramas analizados.

Luego, se procederá a pintar la información de cada fotograma, es decir, una vez tenemos los fotogramas separados del video y la información de la detección de cada fotograma, podemos pintar un cuadro delimitador o *'bounding box'*, usando los puntos y el tipo de objeto que obtenemos en el *'csv'*.

Con este objetivo, se usará el manejo de ficheros que se utilizará para leer de los archivos de extensión *'csv'* la información de las detecciones de los vehículos y se aplicará el método *'rectangle'* de la librería cv2 mencionada anteriormente de la siguiente manera:

```
cv2.rectangle(frame, (x,y), (w,h), (0,255,0), 2)
```

Una vez realizada la pintura de los cuadros delimitadores sobre los fotogramas se guardarán en una carpeta dichas imágenes.

Finalmente se procederá a juntar todos los fotogramas con los que se han trabajado para obtener el video final con las detecciones ya pintadas sobre los vehículos detectados por la Yolov5.



Figura 4.1: Ejemplo de fotograma pintado

## 4.2. Seguimiento de vehículos

En este apartado de la memoria se va a explicar el seguimiento o *'tracking'* de los vehículos, para ello una vez hecha la detección, se ha usado el *Tracker* de la biblioteca *norfair* que se explicó en el apartado anterior. Los argumentos que se deben pasar al *tracker* son el tamaño de la imagen, la confianza con la que debe detectar los vehículos, el identificador de las clases de los objetos que se quiera detectar (por ejemplo, si queremos excluir alguno simplemente no lo ponemos), y la forma de la caja de detección que puede ser una caja delimitadora o *bounding box* que es la que se usará o centroides.

Mediante la función *tracker.update* y pasando como argumento las detecciones, dicha biblioteca devuelve los objetos con el seguimiento realizado y en último paso se vuelve a usar esta librería para dibujar las cajas delimitadoras de las detecciones y las cajas del seguimiento realizado para cada identificador de los vehículos que se han detectado del video de tráfico que se ha insertado.

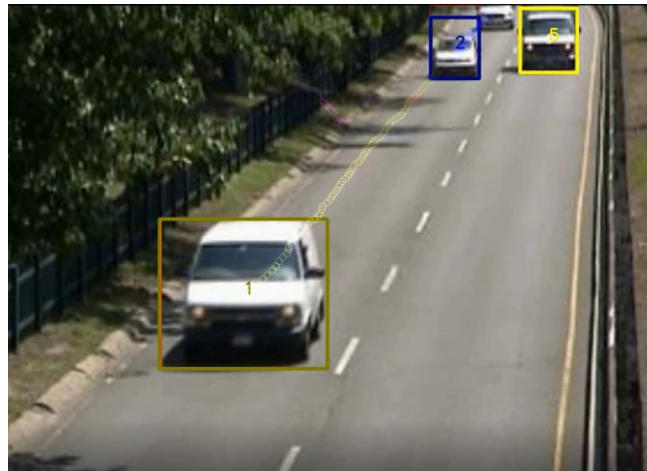


Figura 4.2: Ejemplo de seguimiento de vehículos. Fuente: [29]

### 4.3. Creación de videos

Para la creación de videos se ha usado el programa explicado brevemente en el apartado anterior, *CARLA simulator*, que permite la creación de videos de tráfico con las condiciones que se deseen tanto de escenario como de los vehículos y su comportamiento.

CARLA trae una serie de ejemplos ya creados que se pueden ejecutar mediante la consola que permiten todo tipo de comportamientos, como la generación de tráfico, el cambio de clima, o el piloto manual que es una herramienta interesante para ver como funciona en sí el comportamiento de un vehículo en el simulador.

Todos estos ejemplos y el código que se va a realizar se ejecutarán mediante la consola de Windows una vez abierto el simulador.

Para la generación de los videos se han creado dos programas con la misma estructura en lenguaje Python, pero cambia la posición del actor y algunos detalles más para que no sea exactamente igual que la simulación anterior.

En primer lugar conectamos carla con el cliente, para ello usamos el comando `carla.client` poniendo la dirección 'localhost' y el puerto 2000.

#### 4.3.1. Elección de mapa

En este apartado se va a hablar de la elección del mapa, en primer lugar se procede a crear el mundo para ello obtenemos el mundo con `get_world()` del cliente y cargamos el mapa que se quiera usar, para este proyecto hemos

buscado una autovía para la generación de los videos por ello en este caso se ha usado el Town04 debido a que el objetivo de este proyecto es enfocarse en las cámaras de video vigilancia en un autovía desde un plano horizontal y desde un plano vertical, y este mapa cumple las características que buscamos.

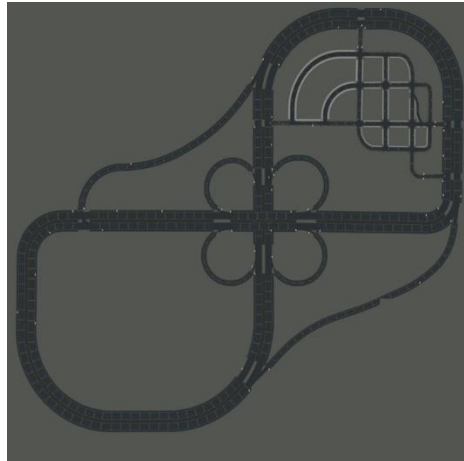


Figura 4.3: Mapa Town04. Fuente: [19]

#### 4.3.2. Generación de actores

CARLA define los actores como cualquier cosa que juega un papel en la simulación o que se puede mover, esto incluye a peatones, vehículos, señales de tráfico o sensores [24].

En el código se obtiene la librería *blueprint library* que es un resumen de todos los actores y sus atributos disponibles para el usuario de CARLA. En nuestro proyecto se usa para filtrar los tipos de vehículos que existen en CARLA. También se hace uso de los *spawn points* del mapa que son una serie de ubicaciones de generación que nos proporciona el mapa con las transformaciones recomendadas para coger una referencia de en qué puntos existe la posibilidad de que aparezcan los vehículos. Lo primero que va a aparecer en el mapa será el espectador, que nos permite ver la circulación de los coches desde un ángulo adecuado para grabar el video, para ello obtenemos el espectador del mundo usando el comando (1) y haremos aparecer el vehículo desde una perspectiva adecuada con el comando (2)

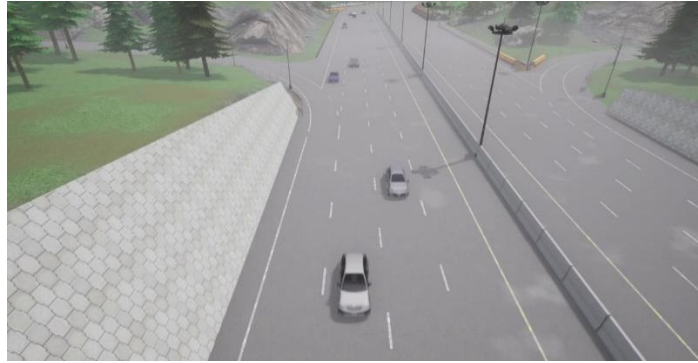
```
world.get_spectator() (1)
```

```
carla.Transform() (2)
```

Finalmente se añadirá a una lista de actores que servirá para guardar todos los actores presentes en el mapa.

```
actor_list.append(npc)
```

Siendo *npc* los actores que se pasaran por argumento a la función de añadir a el array de la lista de actores.



**Figura 4.4: Vista del espectador en Carla**

El siguiente paso es hacer aparecer todos los vehículos que se busca que aparezcan en el video de tráfico que se va a realizar, para ello se usa un bucle “for” hasta el número de vehículos que deseen que aparezcan. Después se ha buscado una ubicación que corresponda con los videos que se quieren realizar, y se ha guardado para todos los vehículos cambiando la coordenada x e y entre un valor aleatorio entre cinco y menos cinco usando el siguiente comando para evitar que intenten aparecer en las mismas coordenadas siempre, y se de variedad de vehículos en distintos carriles.

```
random.randint(-5, 5)
```

Para que no aparezcan algunos vehículos como son las bicicletas (debido a que en una autovía no pueden circular este tipo de vehículos), se ha creado una lista de quince tipos de vehículos que aparecerán en el mapa de una forma aleatoria, y en el caso de que sea un vehículo de dos ruedas se cambiará por un coche normal de esta manera:

```
is_bike = vehicle_bp.get_attribute('number_of_wheels')
if(is_bike.as_int() == 2):
    random.choice(bp_lib.filter('vehicle.lincoln.mkz_2020'))
```



Para hacer aparecer los vehículos se usará el comando:

```
world.try_spawn_actor()
```

Indicando en el argumento el tipo de vehículo y las coordenadas donde queremos que aparezca. En caso de que no pueda aparecer por que en esa misma posición esta circulando otro vehículo se volverá a cambiar la posición de x e y entre +5 y -5 hasta que pueda aparecer el vehículo sin ningún problema.

#### 4.3.3. Comportamiento de los vehículos

A la hora de controlar el comportamiento de los vehículos se ha usado un piloto automático con el comando:

```
set_autopilot(enable=True)
```

Para que el coche tenga un comportamiento automático y lógico. Finalmente lo añadimos a la lista de actores y dormimos con el comando:

```
time.sleep(2.3)
```

Para que aparezcan los vehículos poco a poco en vez de todos seguidos y así prevenir de accidentes justo al aparecer o trayectorias indeseadas, por ello he usado 2.3 segundos que es un tiempo adecuado para que cumpla esas condiciones.

También, se ha añadido que con una probabilidad del 5% (es decir en torno a cinco de cada 100 vehículos) puedan existir trayectorias anómalas en los vehículos, es decir, que el 5% de los vehículos que aparezcan puedan tener una velocidad más alta, una velocidad más lenta notablemente o una trayectoria de zigzag. Para conseguir estas trayectorias anómalas se ha hecho uso de *Traffic Manager* que es el módulo que controla los vehículos en modo de piloto automático en una simulación. Esto permite personalizar algunos comportamientos para establecer circunstancias de aprendizaje específicas [18].

Para ello primero se usa dos comandos como pasos iniciales para obtener los puertos y el traffic manager que son:

```
tm = client.get_trafficmanager(8000)
```

```
tm_port = tm.get_port()
```

Una vez realizados estos comandos, se proceden a introducir los parámetros para provocar las trayectorias anómalas, con el uso del comando:

```
tm.vehicle_percentage_speed_difference(npc, 60)
```

Sabiendo que si se usa un valor mayor que 30 (que es el valor predeterminado) aumentará la velocidad y si se usa un valor menor que cero (negativos) se reducirá la velocidad, de esta forma conseguiremos cambios en sus trayectorias.

Además, se añadira en funcion de si es más lento o más rápido una intención de cambio de carril (los vehículos más rápidos tenderán a circular por el carril izquierdo y los más lentos tenderán a circular por el carril derecho).

```
tm.force_lane_change(npc, True)
```

Si usamos el argumento *True* el vehículo intentará cambiarse siempre que pueda hacia el carril izquierdo, y si usamos *False* en el argumento, los vehículos tenderán a cambiarse al carril derecho siempre que tengan esa posibilidad.

Una vez terminada toda la simulación se usará un *finally* para destruir a todos los actores de la lista de actores mediante un bucle *for* y el comando *destroy* tal y como se indica en las siguientes lineas de código:

```
finally:
    time.sleep(2.3)
    print('Destruyendo actores')
    for actor in actor_list:
        actor.destroy()
    print('Programa terminado.')
```

Un ejemplo de la línea de comandos podemos verla en la Figura 4.5 donde se puede observar:

- Las coordenadas de aparición de los vehículos
- Las coordenadas de aparición del espectador
- El número de vehículos que se busca que aparezcan
- El identificador de cada vehículo que aparece
- Los vehículos anómalos que aparecen y el segundo en el que lo hacen.

```

C:\Users\Manuel\Desktop\ismael\WindowsNoEditor\PythonAPI\examples>python second.py
Mapa cargado: World(id=6694012121931196640)
INFO: Found the required file in cache! Carla/Maps/TM/Town04.bin
Coordenadas de spawn de vehiculos: Transform(Location(x=-13.395881, y=-212.560928, z=0.281942), Rotation(pitch=0.000000, yaw=89.775124, roll=0.000000))
Coordenadas de spawn de espectador: Transform(Location(x=-12.610921, y=-12.562469, z=14.988697), Rotation(pitch=-30.000000, yaw=269.775116, roll=0.000000))
Vehiculos spawnados: 100
0
1
2
3
4
5
6
7
Vehiculo con trayectoria anómala: Velocidad lenta en el segundo: 18.4
8
9
10
11
12
13
14
15
16
17
18
19
20
Vehiculo con trayectoria anómala: Velocidad excesiva en el segundo: 48.3
21

```

Figura 4.5: Línea de comandos de la ejecución de una simulación

Toda esta información que da la línea de comandos puede ser útil a la hora de analizar los resultados para estimar cuándo existirá un vehículo con trayectoria anómala y cuántos vehículos de ese estilo pueden existir.

## 4.4. Estimación de la velocidad

En este apartado se va a explicar la parte de desarrollo del código para conseguir estimar la velocidad relativa a la cámara de objetos en movimiento.

En primer lugar, se aprovecha el momento en el que se realizan las detecciones de la YOLO para calcular el área simplemente restando las coordenadas de  $x_{max}$  –  $x_{min}$  y el mismo procedimiento para el eje Y, esto nos da la base y la altura del rectángulo que aplicando la fórmula del área de un rectángulo (la base multiplicada por la altura) se obtiene el área de el identificador correspondiente al instante de la detección, que se guardará en un *array* de áreas.

Posteriormente se crea un array bidimensional en el que se almacenará para cada identificador y cada fotograma (o instante) en el que aparezca detectado, el área correspondiente a ese vehículo.

El siguiente paso es un bucle que recorra ese *array* para todos los identificadores. Se creará un nuevo *array* para guardar el tiempo en segundos, es decir, el número de fotograma dividido por 30, debido a que se ha usado 30 fotogramas por segundo (así se obtendrá el tiempo en segundos). Este *array*

deberá trasponerse para que encaje en el subprograma de la regresión lineal y se creará un segundo *array* que almacene la inversa de la raíz cuadrada de el área multiplicada por  $\frac{50\pi}{180*N}$ , siendo N el número de píxeles de la resolución horizontal y 50 el ángulo horizontal de la cámara, el valor que le hemos dado es un valor estimado, en caso de que se quieran cálculos más exactos habría que calcular el ángulo de visión de la cámara concreta.

Una vez conseguidos esos dos *arrays* de números reales (*float*) se realizará la regresión lineal siendo la variable independiente el tiempo y la variable dependiente la inversa de la raíz cuadrada del área de cada identificador en un instante concreto mediante el siguiente comando:

```
regr = linear_model.LinearRegression()

regr.fit(diabetes_x_train, diabetes_y_train)

diabetes_y_pred = regr.predict(diabetes_x_test)
```

Una vez tenemos estos parámetros dibujamos las gráficas usando *plot* para ver la regresión lineal, en la Figura 4.6 se puede ver como se daría una recta de regresión lineal tras pasar por el algoritmo, donde el eje X es el tiempo desde que aparece el vehículo hasta que acaba su trayectoria en el video y el eje Y es la inversa de la raíz cuadrada del área de cada fotograma del video.

En la Figura 4.6 cómo se indica en la imagen, los puntos azules serían los datos y la recta roja la recta de regresión lineal.

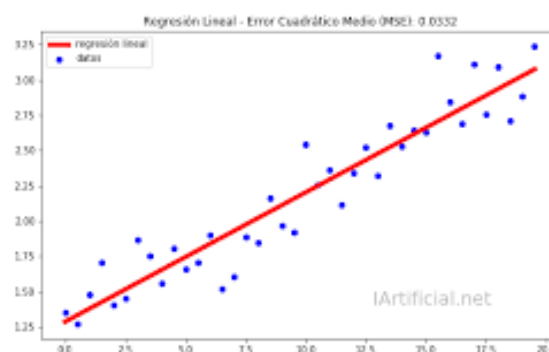


Figura 4.6: Recta de regresión lineal. Fuente: [25]

Finalmente se calcula la velocidad sabiendo que es el diámetro del vehículo multiplicado por el coeficiente de la regresión lineal suponiendo que el diámetro es un valor entre dos y tres.

Se almacenan los valores en un *array* de velocidades y se hace una conversión de la velocidad para pasar de m/s a km/h.

Hay algunos casos en que las velocidades no son almacenadas en ese *array* debido a que son velocidades excesivamente altas a causa de un fallo en el algoritmo de seguimiento o *tracking* así que esos errores del *tracker* no vamos a tenerlo en cuenta a la hora de realizar el análisis de los resultados y simplemente vamos a filtrar esos identificadores con estimaciones erróneas para que no aparezcan.

Este algoritmo también nos permite saber cuáles son los vehículos que poseen una trayectoria anómala, en este caso para ver que vehículos tienen una velocidad por encima de lo normal o por debajo de lo normal.

Tanto el cálculo de las velocidades de los identificadores, cómo los identificadores con trayectorias anómalas y su motivo se han almacenado cada uno en su correspondiente fichero de extensión *.txt* que nos servirá posteriormente para analizar los resultados obtenidos a la hora de estimar la velocidad relativa a la cámara de objetos en movimiento.

## Capítulo 5. Análisis de los resultados

En este capítulo vamos a analizar los resultados obtenidos después de varias simulaciones con videos tanto reales como generados con el simulador CARLA y comparar esos resultados con lo esperado en las condiciones que hemos planteado el proyecto.

A la hora de analizar los resultados de los videos de CARLA se han generado dos videos, uno desde un ángulo vertical y otro desde un ángulo horizontal para probar el seguimiento de los vehículos desde distintas perspectivas y poder tener una colección de casos a analizar con más situaciones.

### 5.1. Videos reales

Ahora se van a utilizar unos videos reales con distinto tráfico vial y distintas velocidades para poder analizar los distintos resultados que obtenemos con condiciones y características diferentes.

En estos video reales la calidad no es muy alta y además hay muchos vehículos con lo que detecta todos los posibles el *tracker*, con lo que los vehículos que no son detectados o los vehículos que tienen algún error en la trayectoria es debido a las condiciones del video y las detecciones del *tracker*.

El primer video real se va a analizar es un video con un tráfico vial medianamente alto, pero sin que los vehículos estén parados o tengan que ir excesivamente lento, todos los vehículos pueden tener un tránsito normal como vemos en la figura 5.1.



Figura 5.1: Video de tráfico de una autopista. Fuente: [29]

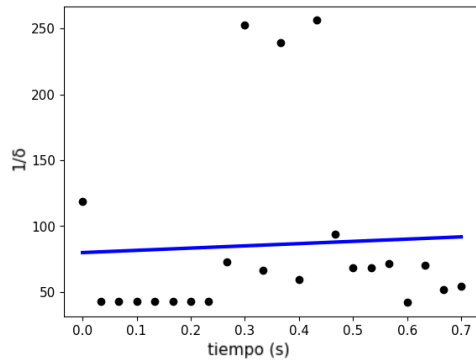
Tras pasar el video de tráfico por el *tracker* obtenemos un video que nos detecta los vehículos y predice su trayectoria tras pasar por el algoritmo de seguimiento como se puede observar en la Figura 5.2. Como vemos es esa figura no todos los vehículos son detectados por los motivos explicados en la introducción a este capítulo.



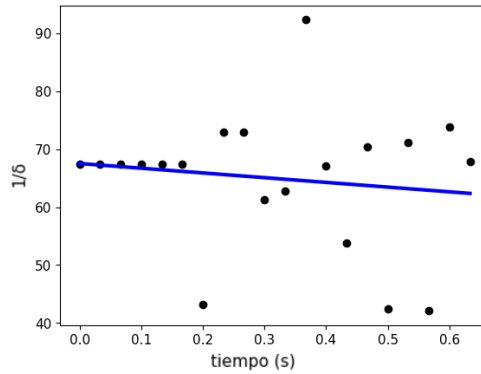
Figura 5.2: Video de tráfico tras pasar por el *tracker*. Fuente: [29]

Los resultados que obtenemos tras pasar el video de tráfico por los algoritmos implementados son que todos los vehículos siguen su trayectoria a una velocidad esperada inicialmente.

Vamos a analizar por ejemplo el vehículo catorce y el dieciséis para ello vamos a ver en las Figuras 5.3a y 5.3b sus rectas de regresión.



**Figura 5.3a: Recta de regresión del  
vehículo catorce**



**Figura 5.3b: Recta de regresión  
del vehículo dieciséis**

El vehículo catorce está en el carril más rápido con lo que su velocidad esperamos que sea alta, y con nuestros cálculos da 122.71 km/h con lo que corresponde con lo esperado.

Por otro lado, el vehículo con identificador dieciséis que se esperaba que tenga una velocidad más baja debido a que está en un carril más ocupado y se obtiene una velocidad de 58.666 km/h, algo que también encajaría con lo esperado.

Dentro de los datos que se reciben sobre las trayectorias anómalas nos indican por ejemplo que los vehículos con identificadores dos y cuatro tienen una velocidad muy lenta, es decir, por debajo de los 20 km/h, es algo normal por que como se ve en la Figura 5.4 el vehículo cuatro (morado) va justo detrás de un camión por lo que es normal que su velocidad sea baja, devuelve como resultado 5.9392 km/h y el vehículo dos (azul) a medida que avanza el video, se queda atrás como se puede ver en la Figura 5.2 que es una figura de unos segundos después y devuelve como resultado 18.434 km/h, que no es algo tan bajo como el vehículo con identificador 4, pero también está por debajo de lo normal.





**Figura 5.4: Fotograma del video de tráfico con seguimiento.**

El segundo video real que se va a analizar es un video con un tráfico denso, los vehículos se encuentran en una retención vial y la velocidad es muy lenta debido a que los vehículos avanzan muy despacio uno detrás de otro sin poder cambiar de carril como se muestra en la Figura 5.5



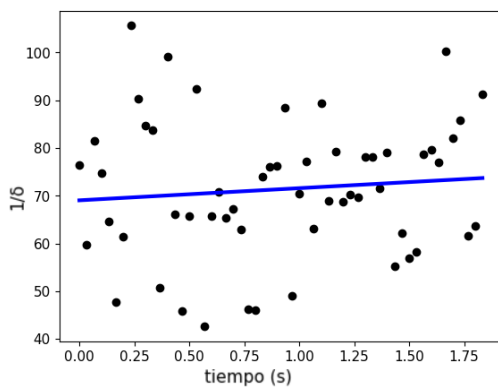
**Figura 5.5: Autovía con tráfico denso. Fuente: [29]**

Los resultados que se esperan obtener son de vehículos con una velocidad más lenta de lo normal ya que como se ha explicado estamos ante un escenario de una retención ya que existen una gran cantidad de vehículos circulando, tras pasar el video por el algoritmo que hemos diseñado obtenemos un video con fotogramas como los de la Figura 5.6

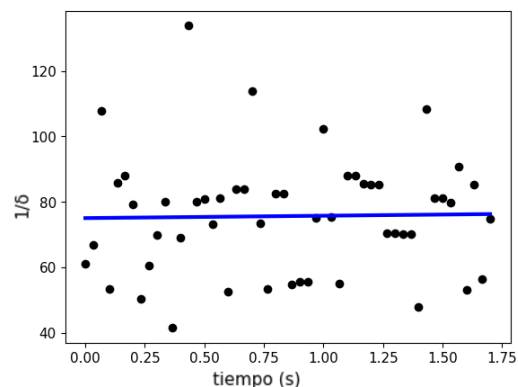


**Figura 5.6: Seguimiento de vehículos en video real**

Ahora se van a analizar los vehículos dos y doce para ello, como se ha hecho anteriormente se vuelve a analizar las rectas de regresión como se ve en la Figuras 5.7a y 5.7b.



**Figura 5.7a: Recta de regresión del vehículo dos**



**Figura 5.7d: Recta de regresión del vehículo doce**

De este video obtenemos lo que esperábamos, que vehículos como el identificador dos (azul) tiene una velocidad de 18.336 km/h que está circulando, pero a una velocidad muy baja o el vehículo con identificador doce (naranja) que está prácticamente parado a lo largo de todo el video que tiene una velocidad de 5.2198 km/h ya que está en el carril derecho que se puede apreciar en la Figura 5.6 que es el carril más ocupado por los vehículos, con lo que encaja con lo que se esperaba que sucediera en este video de tráfico.

Respecto a las trayectorias anómalas en este video de tráfico tenemos que la mayoría de los vehículos van a una velocidad muy lenta por lo que se consideran

como vehículos con trayectorias anómalas a la mayoría de los vehículos identificados.

Finalmente se va a analizar un último video real con más calidad para ver cómo funciona el *tracker* con mejores detecciones, uno de los fotogramas del video que se va a realizar se puede visualizar en la Figura 5.8



Figura 5.8: Fotograma de una autovía concurrida. Fuente: [27], [28]

De este video se generan fotogramas como el que se ve en la Figura 5.9 en el que se ha realizado las detecciones y seguimiento tras pasar por el modelo que se ha realizado.

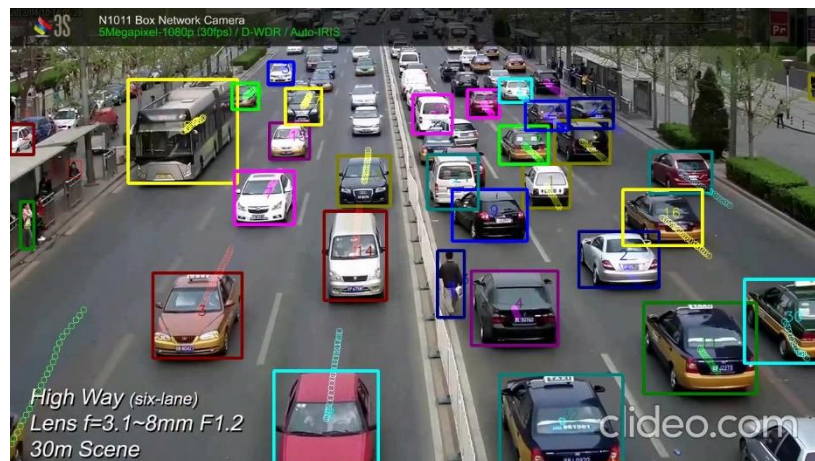
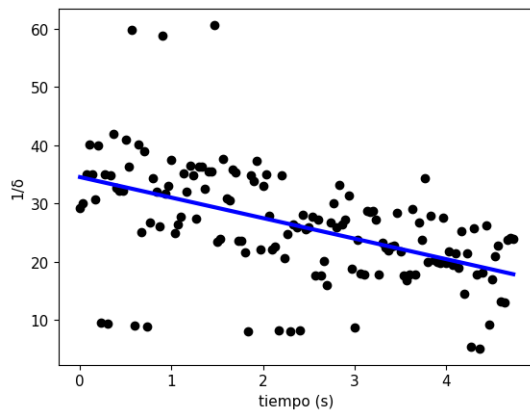


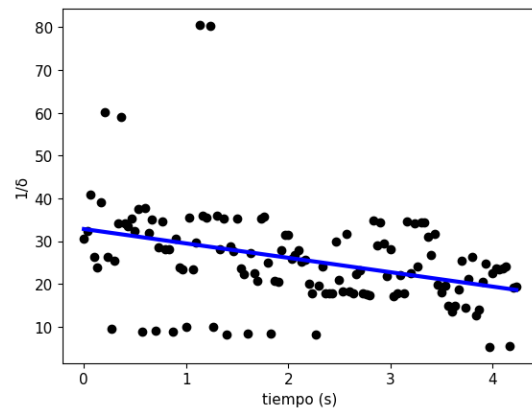
Figura 5.9: Fotograma de un video de tráfico tras pasar por el modelo

Como pasaba en otros videos, ocurren también que algunos vehículos son opacados por otros y no pueden llegar a ser detectados, también hay vehículos que deben acercarse más para que se considere con confianza que son detecciones.

En este video de tráfico las detecciones son mucho más claras debido a la calidad del video y del ángulo de grabación de la cámara de videovigilancia.



**Figura 5.10a: Recta de regresión del  
vehículo 27**



**Figura 5.10b: Recta de regresión del  
vehículo cuatro**

Ahora se van a analizar las regresiones lineales de las figuras 5.10a y 5.10b de dos vehículos como son el vehículo con identificador 27 y el vehículo con identificador cuatro.

El vehículo con identificador 27 tiene una velocidad media de 25.352 km/h, con lo cual es un vehículo con trayectoria normal, ya que este vehículo sigue a lo largo del video una trayectoria constante a una velocidad no muy alta pero no por debajo de lo normal, con lo cual es lógico que su velocidad sea ese valor mencionado, este vehículo circula por un carril transitado, pero no parado.

El siguiente vehículo es el que usa el identificador cuatro tiene una velocidad de 0.39194 km/h, en este caso el vehículo inicialmente en el video estaba en movimiento muy lento, pero al finalizar el video el vehículo acaba parado por el atasco que se produce en el carril en el que circula, por lo que es lógico que su velocidad sea más lenta incluso muy aproximada a 0 km/h.

Respecto a los vehículos anómalos todos son por vehículos lentos y se produce en torno al 55.36% de los vehículos, esta justificado debido a que en el video la mayoría de los vehículos son vehículos que circulan lento debido que existen atascos en ciertos carriles y en los demás hay mucha circulación vial y deben conservar una velocidad moderada.

## 5.2. Videos simulados

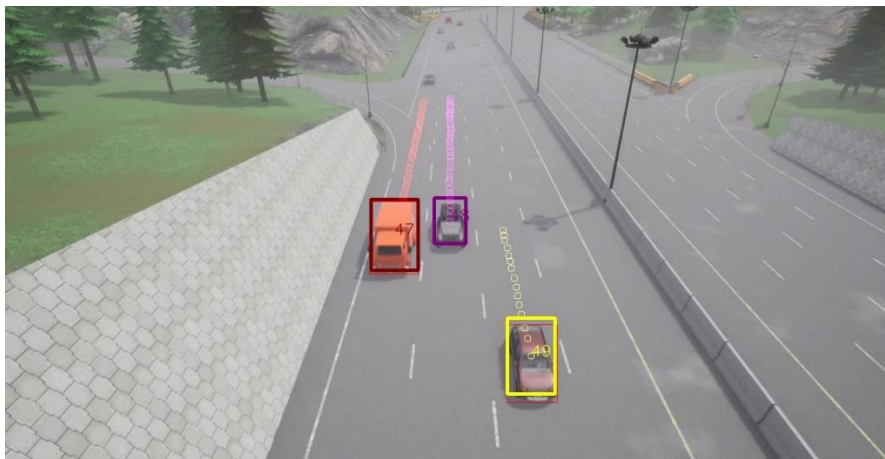
El primer video que se va a analizar es un video simulado desde una perspectiva vertical como se muestra en la Figura 5.11



**Figura 5.11: Simulación de una autovía**

La Figura 5.11 es una autovía desde una perspectiva vertical del espectador, lo normal sería que los vehículos circulen entre 20 y 120 km/h, en la simulación se ven todos los posibles vehículos que pueden circular por la autovía con sus respectivas características.

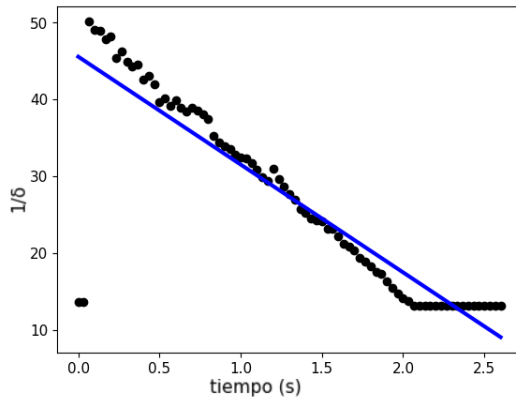
Tras pasar el video por el algoritmo obtenemos fotogramas del estilo de la Figura 5.12 que aparece justo debajo.



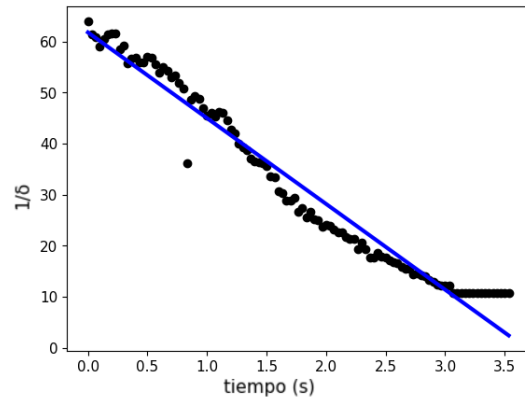
**Figura 5.12: Video de la simulación tras pasar por el *tracker***

Tras pasar por el algoritmo vamos a ver los resultados que se ha obtenido de cada identificador cuándo se ha aplicado la regresión lineal que permite calcular la velocidad en la Figura 5.13a y 5.13b.





**Figura 5.13a: Recta de regresión del  
vehículo trece**



**Figura 5.13a: Recta de regresión del  
vehículo 28**

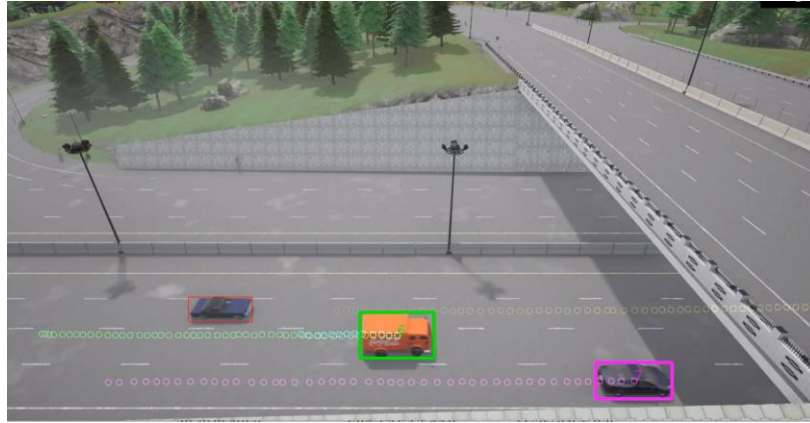
En las Figuras 5.13 hemos cogido la recta de regresión del vehículo trece y 28. Ambos vehículos circulan a una velocidad normal, el vehículo trece a 101.1 km/h y el vehículo 28 a 120.9 km/h aproximadamente, esto significa que son vehículos que cumplen correctamente con la velocidad de la vía.

Respecto a los vehículos con trayectorias anómalas se da el caso de que existen tantos vehículos con velocidad más alta de lo normal y también más baja de lo normal. En torno al 19% son vehículos lentos y otro 19% vehículos rápidos, esto se debe porque, pese a que las simulaciones que realizamos tenían un menor porcentaje de vehículos con trayectorias anómalas, si los vehículos normales no pueden seguir su ritmo debido a que hay vehículos lentos delante, también cogerían dicha trayectoria anómala.



**Figura 5.14: Simulación de una autovía desde un ángulo horizontal**

Para el segundo video hemos cogido una perspectiva horizontal como se ve en la Figura 5.14 para analizar los resultados y tener más posibilidades de estudio.

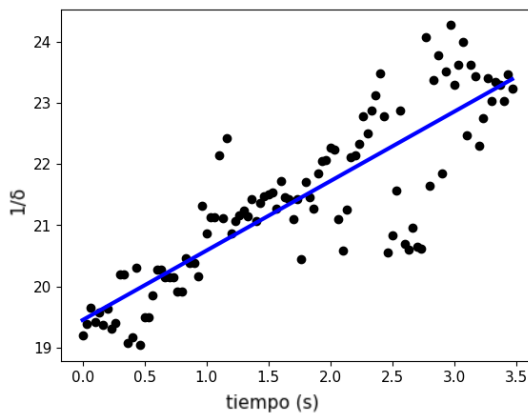


**Figura 5.15: Seguimiento desde una perspectiva horizontal**

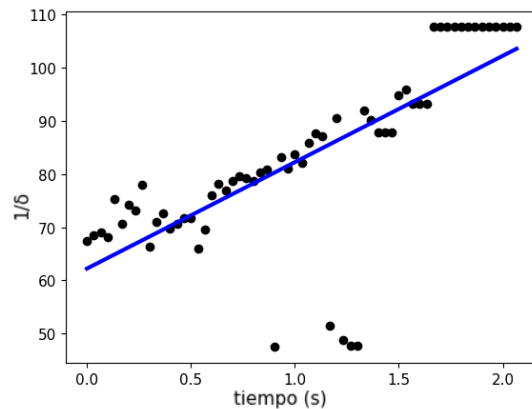
Tras pasar por el algoritmo de seguimiento se obtiene un video con detecciones y predicción de trayectoria como en el fotograma de la Figura 5.15.

En este caso el *tracker* usado de la librería *norfair* tiene ciertos vehículos que no es capaz de detectar. También ocurre que algunos vehículos pueden ser opacados por otros más grandes (como pueden ser camiones) y no se consigue detectar a esos vehículos ya que el *tracker* no consigue verlos de forma adecuada.

Las gráficas generadas respecto a la regresión lineal se ven en las Figuras 5.16a y 5.16b



**Figura 5.16a: Recta de regresión del  
vehículo 79**



**Figura 5.16b: Recta de regresión del  
vehículo 73**

Como se puede ver en las Figuras 5.16a y 5.16b, uno es un vehículo rápido (identificador 73) que circula a unos 143.92 km/h y otro es un vehículo lento (identificador 79) que circula a unos 8.1683 km/h, esto encaja perfectamente ya que el vehículo con identificador 79 es un vehículo con una velocidad muy lenta

y además es una caravana que es un vehículo más lento que otros, con lo que se justifica ese resultado. Por otro lado, el vehículo con identificador 73 es un turismo que circula a gran velocidad por la carretera vertical que hay por encima de la horizontal y sin apenas tráfico vial, con lo que es lógico que tenga una velocidad más alta como es el caso y también queda justificado.





## Capítulo 6. Conclusiones y trabajo futuro

En esta sección se va a explicar las conclusiones que se han sacado de los resultados obtenidos después de todo el proceso realizado y el análisis del código y los posibles trabajos futuros que se pueden implementar en base al código y al proyecto que se ha realizado.

### 6.1. Conclusiones

El objetivo de este proyecto era explicar la tecnología de la detección de vehículos en videos y realizar un análisis detallado de su trayectoria para luego poder centrarnos en la estimación de la velocidad y realizar los experimentos necesarios para comprobar la utilidad y potencial de esta herramienta.

Desde el inicio del trabajo se ha explicado el marco teórico necesario para realizar los algoritmos y los principales escenarios y tareas donde podemos utilizarlo. Para acabar el trabajo se expusieron los resultados obtenidos y se compararon entre ellos explicando la utilidad que tiene y cómo actúa en ciertos contextos.

Después de explicar en los primeros dos capítulos de la memoria los aspectos teóricos, en los siguientes capítulos nos centramos en el desarrollo y el diseño de un sistema donde poder utilizar dichos aspectos teóricos explicados. Basándonos en el comportamiento que usa para las detecciones la YOLOv5 y el algoritmo de seguimiento de la biblioteca *norfair* que permiten el procesamiento de imágenes en tiempo real, hemos podido desarrollar nuestro propio modelo para detección y seguimiento de vehículos y posteriormente la estimación de su velocidad mediante un proceso matemático.

Finalmente, tras analizar los resultados se llegaron a obtener los resultados que esperábamos en diferentes contextos que se ven respaldados en el capítulo cinco ya que la velocidad estimada corresponde a lo que podemos visualizar en los videos de tráfico que hemos usado.

En el ámbito personal, este proyecto me ha ayudado a desarrollar mis conocimientos en el área de la inteligencia artificial y el aprendizaje profundo y aprender un nuevo lenguaje de programación muy útil y popular como es Python, me ha gustado mucho realizar este proyecto y poder acercarme más a este mundo que me parece muy interesante y me supone un gran avance académico.

Para acabar se van a exponer en el siguiente punto algunas posibles modificaciones para mejorar los resultados obtenidos o para llevar a cabo otros proyectos en base a este.

## 6.2. Trabajo futuro

Existen varios trabajos futuros que se pueden plantear para mejorar el sistema implementado. Siguiendo con la línea continuista de nuestro proyecto se podría incluir información y analizar los fotogramas y velocidad en base a el estado de la calzada debido a las condiciones climatológicas, por ejemplo, cuando hay lluvia, niebla o nieve, ya que los vehículos deberían circular con más precaución y menor velocidad, esto nos permitiría llevar la mejora de la seguridad vial a un punto superior y controlar mejor los sucesos que pueden ocurrir en ciertas circunstancias viales. O también aplicar nuestro trabajo a zonas más concurridas por personas y ver como interactúan los peatones y vehículos en un mismo contexto esto nos puede traer grandes beneficios por ejemplo para poder ver accidentes viales como atropellos, la dificultad que tenemos en este aspecto es la calidad de video que se obtenga en esas condiciones climatológicas nos causaría un problema a la hora de la confianza en las detecciones.

Del mismo modo se podrían realizar experimentos con nuevos detectores (MobileNet o CenterNet) y nuevas redes (ResNet o GoogleNet) y realizar una comparación con nuestros resultados obtenidos para comprobar cuál funciona de mejor manera, pese a que habría que analizar todos los aspectos posibles de cada detector y pensar cuáles son los aspectos más importantes en los que

debemos fijarnos (que cambiarán dependiendo del contexto) para elegir cuál es mejor.

Y como última propuesta se plantea la optimización del código para que sea más eficaz y rápido o usar ordenadores con mayor número de GPU y más potentes para intentar buscar un rendimiento mayor, la dificultad que tiene esta medida es que el presupuesto que supondría para llevar a cabo el modelo es mucho mayor al requerir mayor capacidad de GPU.

## Apéndice A. Uso del sistema

En este apéndice se va a explicar una pequeña introducción a los entornos con los que se han trabajado durante el proyecto. Los videos realizados, códigos realizados y sus resultados están en el siguiente [enlace](#) de drive.

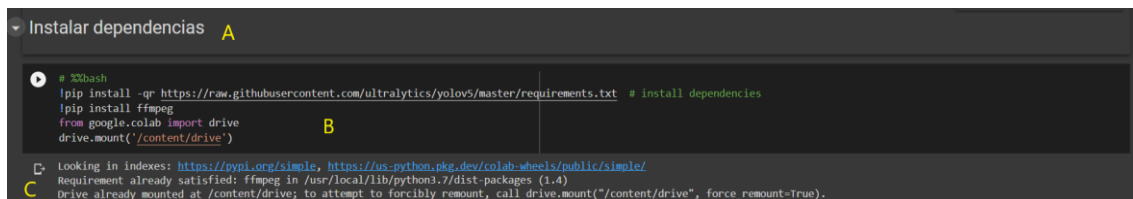
### A.1. Introducción a Google Colab

A fin de hacer una breve introducción al entorno de Google Colab se requiere explicar cuáles son los pasos iniciales a esta herramienta.

Inicialmente se debe crear un cuaderno en blanco (o usar uno ya realizado) y cambiar dentro del apartado “Entorno de ejecución” el tipo de entorno de ejecución para usar la GPU que nos permite usar Google.

A partir de ahí ya podemos instalar las dependencias necesarias que necesitemos para nuestra programación en Python y simplemente ya podemos empezar a ejecutar cada celda paso a paso o ejecutar todas las celdas seguidas y ver cómo avanza nuestro código apartado a apartado.

También da la posibilidad de entre trozos de código escribir fragmentos de texto o incluso imágenes para poder explicar el trozo de código que se va a realizar y ejecutar posteriormente en el cuaderno.



```
Instalar dependencias A
# %%bash
!pip install -qr https://raw.githubusercontent.com/ultralytics/yolov5/master/requirements.txt # install dependencies
!pip install ffmpeg
from google.colab import drive
drive.mount('/content/drive') B
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: ffmpeg in /usr/local/lib/python3.7/dist-packages (1.4)
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True). C
```

Figura A.1: Ejemplo de código en Google Colab

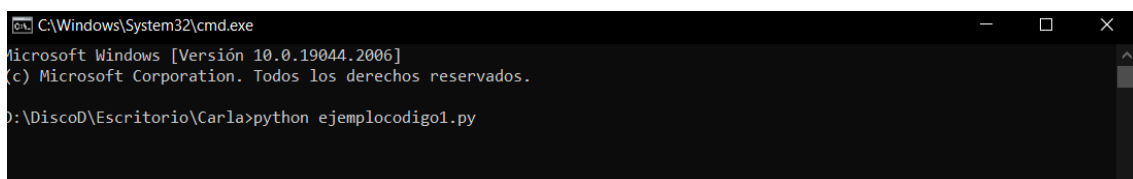
Cómo se puede ver en la Figura A.1, se ha dividido en tres secciones (en amarillo). “A” donde se ha escrito el título de lo que se va a hacer en ese trozo de código, “B” que es el lugar donde se ha realizado el trozo de código correspondiente y “C” que son los resultados que nos da tras ejecutar el código.

## A.1. Introducción a CARLA Simulator

Para introducir un poco a CARLA se va a explicar cuál es la manera de usarlo, en primer lugar, se debe en primer lugar instalar una de las versiones del programa en la página web oficial de el simulador ya que es un software gratuito de libre uso.

En segundo lugar, es necesario abrir el simulador y se visualizará un mundo, pero aún sin vehículos y con un mapa por defecto. Después debemos crear un archivo de extensión de Python con un código que pueda ser compilado y usado o en su defecto usar los códigos de ejemplo que nos trae la carpeta en la que se ha instalado el simulador, que son de gran utilidad para poder empezar a entender el código y el funcionamiento del programa, ya que da la posibilidad de poner llamar a cada archivo de uno en uno en la línea de comandos e ir viendo que función tiene cada uno dentro del simulador.

Para poder abrir y ejecutar los programas realizados lo debemos hacer desde la línea de comandos de Windows tal y como se indica en la Figura A.2



**Figura A.2: Ejemplo de ejecución de código en CARLA**

Y finalmente se volverá a abrir el simulador para visualizar como avanza y se desarrolla el escenario creado mediante el lenguaje de programación Python.

En definitiva, es muy sencillo de usar y tiene una gran colección de códigos de ejemplo para poder introducirnos dentro de la programación.

## Capítulo 7. Referencias

- [1] R. Salas, «Redes neuronales artificiales,» Universidad de Valparaíso. Departamento de Computación, 2004.
- [2] A. & M. E. Nacelle, «Redes neuronales artificiales. Núcleo de ingeniería biomédica,» Universidad de la Republica Uruguay., Uruguay, 2009.
- [3] HostDime, «HostDime,» [En línea]. Available: <https://www.hostdime.com.ar/blog/por-que-son-necesarias-las-gpu-para-entrenar-modelos-de-aprendizaje-profundo/>. [Último acceso: 04 09 2022].
- [4] P. Loncomilla, «Deep learning: Redes convolucionales,» Recuperado de <https://ccc.inaoep.mx/~pgomez/deep/presentations.>, 2016.
- [5] J. Barrios, «JuanBarrios,» [En línea]. Available: <https://www.juanbarrios.com/redes-neurales-convolucionales/>. [Último acceso: 05 09 2022].
- [6] L. Claims, «Lisa Claims,» [En línea]. Available: <https://www.lisainsurtech.com/es/2021/11/23/hablemos-de-la-deteccion-de-objetos/>. [Último acceso: 05 09 2022].

- [7] «Aprende Machine Learning. ¿Cómo funcionan las Convolutional Neural Networks?,» [En línea]. Available: <https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>. [Último acceso: 05 09 2022].
- [8] U. Michelucchi, Advanced Applied Deep Learning: Convolutional Neural Networks, 2019.
- [9] CampusMvp, «CampusMvp,» [En línea]. Available: <https://www.campusmvp.es/recursos/post/los-4-mejores-lenguajes-de-programacion-para-inteligencia-artificial-machine-learning.aspx>. [Último acceso: 2022 09 10].
- [10] A. c. alf, «Aprende con alf,» [En línea]. Available: <https://aprendeconalf.es/docencia/python/manual/pandas/>. [Último acceso: 2022 09 10].
- [11] G. Dreyfus, Neural Network: An Overview, 2005.
- [12] Y. &. R. D. Chauvin, Backpropagation. Theory, Architectures and Applications, 1995.
- [13] khanacademy, «khanacademy,» [En línea]. Available: <https://es.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/optimizing-multivariable-functions/a/what-is-gradient-descent#:~:text=El%20descenso%20de%20gradiente%20es%20un%20algoritmo%20que%20estima%20numéricamente,0%20>. [Último acceso: 10 09 2022].
- [14] Tutorialspoint, «Tutorialspoint,» [En línea]. Available: [https://www.tutorialspoint.com/google\\_colab/what\\_is\\_google\\_colab.htm](https://www.tutorialspoint.com/google_colab/what_is_google_colab.htm). [Último acceso: 11 09 2022].
- [15] J. R. y. A. Farhadi, YOLOv3: An incremental improvement, arXiv, 2018.



- [16] M. Gonzáles, «Wondershare,» [En línea]. Available: <https://filmora.wondershare.es/video-editor/free-video-editor-ffmpeg.html>. [Último acceso: 14 09 2022].
- [17] J. Alori, «Zenodo,» [En línea]. Available: <https://zenodo.org/record/5146254#.YyMPoGxBxPY>. [Último acceso: 15 09 2022].
- [18] Carla, «Carla,» [En línea]. Available: [https://carla.readthedocs.io/en/latest/adv\\_traffic\\_manager/#what-is-the-traffic-manager](https://carla.readthedocs.io/en/latest/adv_traffic_manager/#what-is-the-traffic-manager). [Último acceso: 16 09 2022].
- [19] Carla, «Carla,» [En línea]. Available: [https://carla.readthedocs.io/en/latest/core\\_map/](https://carla.readthedocs.io/en/latest/core_map/). [Último acceso: 16 09 2022].
- [20] «Deci,» [En línea]. Available: <https://deci.ai/blog/how-to-double-yolov5-performance-in-15-minutes/>. [Último acceso: 2022 09 16].
- [21] «Programacion,» [En línea]. Available: [https://programacion.net/articulo/los\\_pickles\\_de\\_python\\_1860](https://programacion.net/articulo/los_pickles_de_python_1860). [Último acceso: 16 09 2022].
- [22] «Grupo US,» [En línea]. Available: <http://grupo.us.es/gtocom/pid/pid10/RedesNeuronales.htm>. [Último acceso: 16 09 2022].
- [23] «Unrealengine,» [En línea]. Available: <https://www.unrealengine.com/en-US/spotlights/carla-democratizes-autonomous-vehicle-r-d-with-free-open-source-simulator>. [Último acceso: 16 09 2022].
- [24] «Carla,» [En línea]. Available: [https://carla.readthedocs.io/en/latest/bp\\_library/](https://carla.readthedocs.io/en/latest/bp_library/). [Último acceso: 16 09 2022].
- [25] «Iartificial,» [En línea]. Available: <https://www.iartificial.net/regresion-lineal-con-ejemplos-en-python/>. [Último acceso: 17 09 2022].

- [26] «OBS project,» [En línea]. Available: <https://obsproject.com/es>. [Último acceso: 18 09 2022].
- [27] «Clideo,» [En línea]. Available: [clideo.com](https://clideo.com). [Último acceso: 19 09 2022].
- [28] «Medium,» [En línea]. Available: <https://medium.com/geekculture/large-scale-object-detection-tracking-with-yolov5-package-c8eca66b80b6>. [Último acceso: 19 09 2022].
- [29] «Pytorch,» [En línea]. Available: [https://pytorch.org/hub/ultralytics\\_yolov5/](https://pytorch.org/hub/ultralytics_yolov5/). [Último acceso: 05 04 2022].