



UNIVERSIDAD DE MÁLAGA



E.T.S. INGENIERÍA  
**INFORMÁTICA**  
UNIVERSIDAD DE MÁLAGA

## Graduado en Ingeniería Informática

Seguimiento de vehículos a través de videovigilancia y estimación de su  
velocidad

Vehicles tracking through video surveillance and estimation of their speed.

Realizado por  
Daniel Lechuga Ruiz

Tutorizado por  
Ezequiel López Rubio

Departamento  
Lenguajes y Ciencia de la Computación  
UNIVERSIDAD DE MÁLAGA

MÁLAGA, septiembre 2023



# Resumen

En la presente investigación, se aborda la problemática de la seguridad vial, centrándose en la detección de vehículos en carretera a través de una cámara montada en un automóvil en movimiento. El principal objetivo es proporcionar una herramienta que permita no solo identificar vehículos cercanos, sino también calcular la distancia a la que se encuentran y estimar su velocidad. De esta manera, el sistema puede alertar al conductor si un vehículo está demasiado cerca, potencialmente previniendo accidentes o permitiendo una reacción temprana ante situaciones imprevistas.

En el enfoque de nuestra investigación, el análisis se centró específicamente en un único vehículo en la carretera: aquel que podría proporcionar la información más relevante en tiempo real. La selección de este vehículo se realiza a través de un filtro que identifica el coche más cercano al centro de la imagen, lo que probablemente corresponde al automóvil directamente frente al conductor en su carril. No obstante, se ha implementado un mecanismo adicional para descartar vehículos que, aunque estén en el centro, se encuentren demasiado lejos, optando en su lugar por vehículos en carriles adyacentes que estén más próximos.

Para lograr esto, se empleó la tecnología YOLOv8, conocida por su eficiencia y precisión en la detección de objetos. A través de este modelo, es posible identificar vehículos, demarcándolos con un cuadro delimitador y asignando un ID único para cada uno. Esta información, combinada con algoritmos de cálculo de distancia y velocidad, brinda una herramienta robusta y en tiempo real para la prevención de incidentes en la carretera.

Todo el sistema fue desarrollado utilizando el lenguaje de programación Python. Para verificar la eficacia de nuestra metodología, se utilizó la fuente de investigación de <https://once-for-auto-driving.github.io/>, la cual proporcionó vídeos e información real de los coches, incluyendo distancia al centro del vehículo y valores del cuadro delimitador. Gracias a estos datos, pudimos

contrastar y validar la precisión de nuestras estimaciones con datos reales, permitiendo una evaluación rigurosa y objetiva de la solución propuesta.

Palabras Claves: Redes neuronales convolucionales, YOLO, vehículo, distancia, caja delimitadora, ID, velocidad

## Abstract

In the present research, the issue of road safety is addressed, focusing on the detection of vehicles on the road through a camera mounted on a moving automobile. The primary aim is to provide a tool that allows for not only the identification of nearby vehicles but also the calculation of their distance and estimation of their speed. In this way, the system can alert the driver if a vehicle is too close, potentially preventing accidents or allowing for early reaction to unforeseen situations.

In our research approach, the analysis specifically targeted a single vehicle on the road: the one that could provide the most relevant real-time information. The selection of this vehicle is done through a filter that identifies the car closest to the center of the image, which likely corresponds to the automobile directly in front of the driver in their lane. Nevertheless, an additional mechanism has been implemented to discard vehicles that, although centered, are too far away, opting instead for vehicles in adjacent lanes that are closer.

To achieve this, YOLOv8 technology was employed, known for its efficiency and accuracy in object detection. Through this model, it is possible to identify vehicles, marking them with a bounding box and assigning a unique ID for each one. This information, combined with distance and speed calculation algorithms, provides a robust and real-time tool for the prevention of road incidents.

The entire system was developed using the Python programming language. To verify the efficacy of our methodology, the research source from <https://oncefor-auto-driving.github.io/> was used, which provided videos and real information about the cars, including distance to the vehicle's center and bounding box values. Thanks to this data, we were able to compare and validate the accuracy of our estimates with real data, allowing for a rigorous and objective evaluation of the proposed solution.

Keywords: Convolutional Neural Networks, YOLO, vehicle, distance, bounding box, ID, speed

# Índice

<b>1.Introducción</b>	<b>9</b>
1.1 Motivación	9
1.2 Estado del arte	10
1.3 Propuesta	11
1.4 Estructura del Proyecto	11
<b>2.Marco Teórico</b>	<b>13</b>
2.1 Detección de objetos y técnicas de visión por computador	13
2.1.1 Historia breve de la visión por computador	13
2.1.2 Relevancia en la detección de objetos	14
2.2 Conceptos básicos de la detección de objetos	15
2.2.1 Caja delimitadora (Bounding box)	15
2.2.2 Identificador (ID) del objeto	15
2.3 Técnicas de estimación de distancia	16
2.4 Redes Neuronales Convolucionales (CNN) en Detección de Vehículos	18
2.4.1 YOLO	22
<b>3.Desarrollo</b>	<b>25</b>
3.1 Obtención y preprocesamiento de datos	25
3.2 Detección de vehículos con YOLO	28
3.3 Estimación de distancia y vehículo seleccionado	30
3.4 Cálculo de diferencia de velocidad	32
3.5 Interfaz gráfica	32
3.6 Optimización y desafíos	33
<b>4. Resultados</b>	<b>37</b>
4.1 Distancia	37
4.1.1 Distancia cercana	37
4.1.2 Distancia mediana	40
4.1.3 Distancia lejana	41
4.2 Diferencia de velocidad	44
4.3 Tipo de vehículo	45
4.3.1 Coche	45
4.3.2 Furgoneta	47
4.3.3 Camión	47
4.4 Visibilidad	49
<b>5. Conclusión</b>	<b>51</b>





# 1. Introducción

## 1.1 Motivación

Hoy en día, una de las principales causas de lesiones y muertes a nivel mundial son los accidentes de tráfico, eventos impredecibles que pueden suceder en cualquier lugar y momento. Por ello gracias a la revolución tecnológica que vivimos hoy en día podemos analizar a través de datos relevantes movimientos en la carretera que nos permiten crear modelos cada vez más exactos para prevenir estas causas y reducir el número de lesiones y muertes.

La seguridad vial en España sigue siendo un tema de suma importancia y preocupación. Según datos recientes, en el año 2022 se produjeron 1.042 siniestros mortales en las carreteras españolas. Estos trágicos eventos dieron lugar a la lamentable pérdida de 1.145 vidas y dejaron a otras 4.008 personas con heridas de gravedad. Al comparar estas cifras con las del año 2019, que se toma como referencia por ser previo a la pandemia del COVID-19, se observa un aumento del 4% en el número de fallecidos, lo que se traduce en 44 personas más. Sin embargo, hay una disminución del 10% en el número de heridos graves, es decir, 425 personas menos que en 2019 [1].

A pesar de las fluctuaciones en las cifras, cada vida perdida o afectada en las carreteras es un recordatorio contundente de la urgente necesidad de mejorar la seguridad vial. Es en este contexto donde los sistemas avanzados de asistencia al conductor y detección de anomalías en carretera adquieren una relevancia crucial. Estas herramientas no solo buscan reducir la incidencia de accidentes, sino también brindar información vital en tiempo real que pueda auxiliar a los conductores en situaciones críticas, permitiéndoles tomar decisiones informadas en fracciones de segundo.

## 1.2 Estado del arte

La movilidad ha experimentado una transformación sin precedentes en las últimas décadas. A medida que avanzamos hacia un futuro donde los vehículos autónomos podrían convertirse en una norma, la seguridad y la precisión en la detección de vehículos y otros objetos en la carretera se han convertido en aspectos cruciales. La detección de vehículos es esencial para garantizar una conducción segura y eficiente, ya sea para un humano al volante o para un sistema autónomo.

Históricamente, las técnicas basadas en visión por computador, como la detección de bordes, el análisis de histogramas de gradientes orientados (HOG) y los clasificadores en cascada, fueron ampliamente utilizadas para la detección de vehículos. Estos métodos, aunque efectivos en ciertas condiciones, a menudo sufrían de falsos positivos y carecían de robustez en condiciones de iluminación variables o en escenarios con vehículos parcialmente ocluidos.

Con el surgimiento de las redes neuronales convolucionales (CNN) y el crecimiento masivo en capacidades de procesamiento y almacenamiento de datos, la detección de vehículos experimentó un cambio de paradigma. Modelos como Faster R-CNN, Single Shot MultiBox Detector (SSD) y YOLO no solo demostraron ser más precisos, sino que también podían detectar vehículos en tiempo real, lo que es crucial para aplicaciones en vehículos autónomos.

La disponibilidad de grandes conjuntos de datos, como COCO, ONCE, KITTI y Waymo Open Dataset, ha impulsado aún más la investigación en este campo. Estos datasets proporcionan imágenes y vídeos etiquetados que permiten a los investigadores entrenar y evaluar sus modelos en escenarios de la vida real.

A medida que avanzamos, la integración de múltiples sensores, como cámaras, LiDAR y radar, junto con algoritmos de fusión de datos, podría ofrecer soluciones más robustas. Además, la adaptabilidad y el aprendizaje continuo en el vehículo, para adaptarse a nuevas condiciones y escenarios, también serán esenciales.

## **1.3 Propuesta**

Nuestra propuesta se centra en detectar la distancia y diferencia de velocidad de un vehículo seleccionado en carretera, tomando como referencia el vehículo que está conduciendo. Inicialmente, la información capturada por la cámara instalada en el vehículo se procesa para detectar vehículos en la carretera. Cada vehículo detectado es analizado y en cada frame del vídeo se escoge un solo vehículo que nos proporciona más información o es más relevante para nuestro análisis. A partir de aquí calculamos la distancia real y la diferencia de velocidad.

## **1.4 Estructura del Proyecto**

Este capítulo sirve como punto de partida, ofreciendo una panorámica general del estado del arte en la detección de vehículos en carretera y estableciendo el contexto en el que se inserta este proyecto, luego en el marco teórico se exploran los conceptos fundamentales de la detección de objetos y las técnicas de visión por computador. Se desglosan las fórmulas utilizadas para la estimación de distancia y se explican en detalle las bases teóricas detrás de las redes neuronales convolucionales y la tecnología YOLO, se pasa al desarrollo en donde se explica la obtención y preprocesamiento de datos, la integración de YOLO con Python, selección del vehículo en cada fotograma, interfaz gráfica y las optimizaciones y desafíos surgidos en el proyecto, luego se analizan los resultados obtenidos y se termina con una conclusión.



## 2. Marco Teórico

Este capítulo se adentra en los fundamentos teóricos que sustentan el desarrollo y comprensión de nuestro proyecto. Primero, exploraremos el fascinante mundo de la visión por computador, trazando su evolución y significado en el contexto de la detección de objetos. A medida que avanzamos, desglosaremos los conceptos esenciales que rigen estas técnicas y nos sumergiremos en las metodologías específicas empleadas para estimar distancias en imágenes. Finalmente, introduciremos las redes neuronales convolucionales (CNN), enfatizando su importancia en la detección moderna, y culminaremos con una discusión sobre YOLO, una de las tecnologías clave que impulsan nuestro sistema.

### 2.1 Detección de objetos y Técnicas de Visión por Computador

#### 2.1.1 Historia breve de la visión por computador

La visión por computador es un campo interdisciplinario que se ha desarrollado con el objetivo de permitir que las máquinas "vean" e interpreten el mundo visual de una manera similar a como lo hacen los humanos. Su origen se remonta a los años 60, cuando se realizaron los primeros experimentos para hacer que las computadoras reconocieran patrones simples y formas geométricas.

**Años 60-70:** Durante estos años, se realizaron los primeros intentos para analizar imágenes digitales y reconocer formas y patrones. Uno de los proyectos pioneros fue el sistema de Larry Roberts en 1963, que reconoció objetos 3D a partir de un conjunto de puntos. En esta década, también se establecieron las bases para el procesamiento de imágenes con operaciones como el filtrado y la detección de bordes. [2]

**Años 80:** Esta década vio el surgimiento de enfoques basados en características y la introducción de técnicas de reconocimiento basadas en el aprendizaje automático.

Se desarrollaron algoritmos para la extracción de características y la correspondencia de puntos entre diferentes imágenes. [3]

**Años 90:** La visión por computador comenzó a tener aplicaciones comerciales, desde la inspección industrial hasta la vigilancia. Las técnicas de aprendizaje profundo empezaron a tomar forma, aunque todavía no dominaban el campo. [4]

**Años 2000 en adelante:** Con la llegada de grandes conjuntos de datos y el aumento de la potencia computacional, las técnicas de aprendizaje profundo comenzaron a dominar el campo de la visión por computador. En 2012, AlexNet, una red neuronal convolucional (CNN), ganó por un amplio margen el desafío de clasificación de imágenes ImageNet, marcando un punto de inflexión en la detección y clasificación de objetos. [5]

## 2.1.2 Relevancia en la detección de objetos

La detección de objetos es una de las tareas fundamentales de la visión por computador y ha sido el foco de muchos avances en las últimas décadas. El objetivo es identificar y localizar objetos específicos dentro de una imagen. Gracias a la visión por computador, es posible:

**Automatizar procesos:** Por ejemplo, en la industria manufacturera, donde las máquinas pueden detectar defectos en productos en tiempo real.

**Mejorar la seguridad:** Como en el proyecto, donde la detección de vehículos puede prevenir accidentes.

**Desarrollar nuevas tecnologías:** Como los coches autónomos, que dependen en gran medida de la capacidad de detectar y comprender su entorno.

La capacidad de las máquinas para "ver" y "entender" imágenes y videos ha abierto puertas a innumerables aplicaciones en diversos campos, desde la medicina hasta el entretenimiento, y seguirá siendo un área de investigación fundamental en el futuro.

## **2.2 Conceptos básicos de la detección de objetos**

### **2.2.1 Caja delimitadora (Bounding box)**

Una bounding box, o caja delimitadora, es un término utilizado en visión por computador y procesamiento de imágenes para describir un rectángulo superpuesto a una imagen que delimita o encierra un objeto de interés dentro de la imagen. Estas cajas son esenciales en la tarea de detección de objetos porque no solo identifican la presencia de un objeto, sino también su posición y escala dentro de la imagen.

Una bounding box se define comúnmente mediante las coordenadas de la esquina superior izquierda ( $x, y$ ) y las dimensiones de la caja: ancho ( $w$ ) y alto ( $h$ ). En el contexto de nuestro proyecto, se define mediante cuatro puntos especificados en la imagen:  $x_1, x_2, y_1, y_2$ . Aquí  $x_1$  e  $x_2$  representan coordenadas en el plano horizontal, mientras que  $y_1$  e  $y_2$  se refieren a coordenadas en el plano vertical. En lugar de definir la caja por su esquina superior izquierda y sus dimensiones, nosotros utilizamos estos cuatro puntos para determinar la posición y tamaño exactos de la caja en la imagen.

### **2.2.2 Identificador (ID) de objeto**

Un identificador (ID) de objeto es un número o etiqueta única asignada a un objeto detectado dentro de una imagen o secuencia de imágenes (como un video). Este identificador permite seguir el objeto a lo largo del tiempo y es esencial para tareas como el seguimiento de objetos en secuencias de video.

Por ejemplo, en un escenario de seguimiento de vehículos en una carretera, si un coche se detecta en varios fotogramas consecutivos, se le asignaría un ID único para garantizar que se reconozca como el mismo coche a lo largo de la secuencia.

Esto es especialmente útil para determinar trayectorias, calcular velocidades y detectar patrones o comportamientos anómalos de los vehículos u objetos en movimiento.

En resumen, mientras que la bounding box proporciona información sobre la ubicación y escala del objeto, el ID de objeto proporciona una forma consistente de identificar y seguir ese objeto a lo largo del tiempo o entre diferentes imágenes o fotogramas.

## 2.3 Técnicas de Estimación de Distancia

**Objetivo:** Estimar la distancia real desde la cámara posicionada en el vehículo de referencia hasta el vehículo seleccionado, utilizando mediciones y parámetros conocidos.

La información estipulada en este apartado viene de la fuente [6].

### Requisitos previos:

1. **Dimensiones del objeto:** Es necesario conocer las medidas reales del objeto, tanto la anchura como altura.
2. **Posicionamiento del objeto:** El objeto debe estar en un plano perpendicular al eje óptico de la cámara, es decir, la línea imaginaria que se proyecta desde el objetivo de la cámara.
3. **Parámetros de la cámara:** Es crucial tener información sobre la distancia focal, así como la anchura y altura del sensor de la cámara.
4. **Medición en la imagen:** Se debe medir las dimensiones aparentes del objeto en la imagen, es decir, en píxeles.
5. **Estabilidad en la focal:** Asegurarse de que el autofocus no modifique de forma significativa la distancia focal. [7].



### Metodología de cálculo:

- **Ángulo de Visión:** Este ángulo describe el campo visual de una cámara en una dirección específica, ya sea horizontal o vertical. Se calcula mediante la fórmula:

$\alpha = 2 * \arctan(s/(2f))$  Aquí,  $s$  es el tamaño del sensor en la dirección seleccionada y  $f$  representa la distancia focal de la cámara.

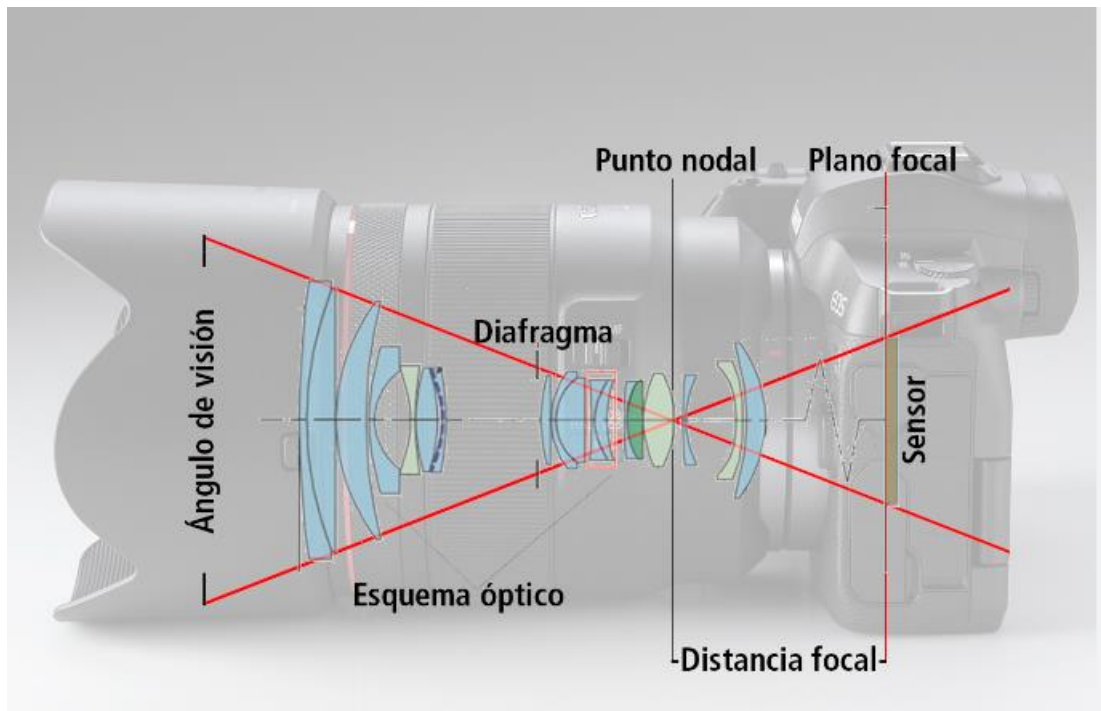


Figura 1: Ángulo de visión en cámara [8]

- **Ángulo por Píxel:** Representa el ángulo que corresponde a un píxel individual en una dirección determinada. Se obtiene al dividir el ángulo de visión en dicha dirección entre el número total de píxeles en esa misma dirección.
- **Diámetro Angular:** Describe el ángulo bajo el cual se observa un objeto desde una posición particular. Su cálculo es:  $\delta = 2 * \arctan(d/(2D))$  Donde  $d$  se refiere al tamaño real del objeto y  $D$  a la distancia entre la cámara y el objeto. [9].

Usando la fórmula del diámetro angular, es posible deducir la relación:

$D = d/(2 * \tan(\delta/2))$  Con esta expresión, se puede estimar la distancia real  $D$  desde el

objeto a la cámara utilizando las mediciones disponibles y los parámetros preestablecidos de la cámara.

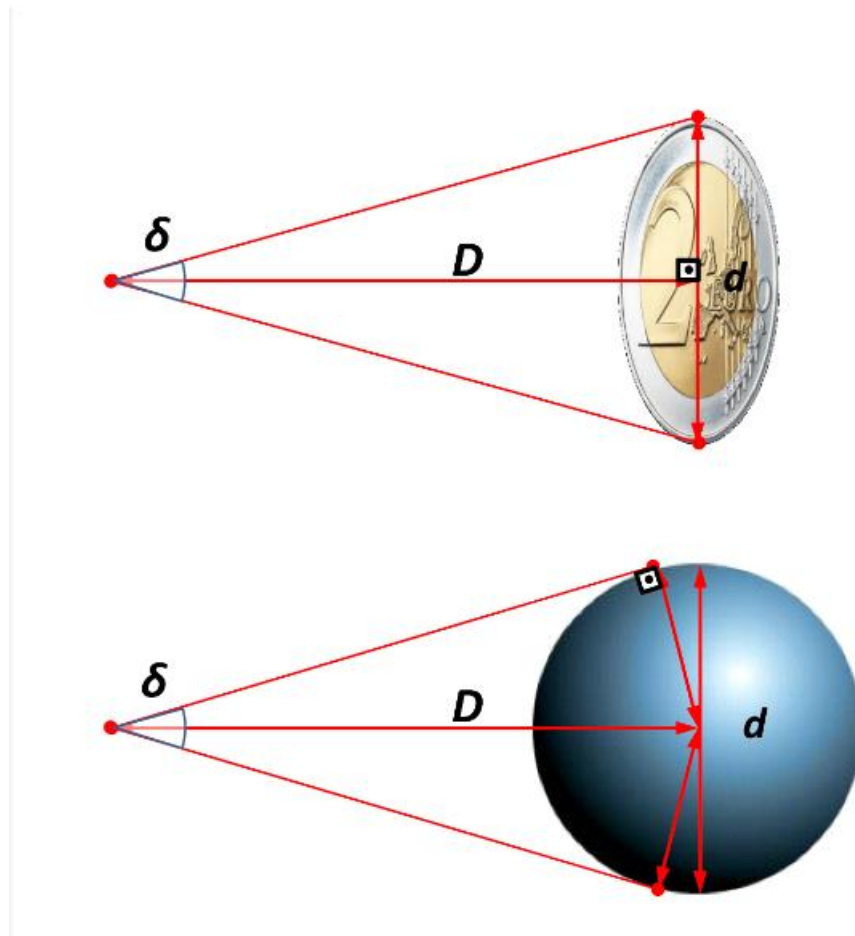


Figura 2: Ejemplo cálculo diámetro angular [10].

Esta distancia  $D$  sería la distancia calculada desde el vehículo de referencia al vehículo seleccionado.

## 2.4 Redes Neuronales Convolucionales (CNN) en Detección de Vehículos

Las **Redes Neuronales Convolucionales** (**CNN**, por sus siglas en inglés de "Convolutional Neural Networks") son una parte fundamental del proyecto. Pertenecen a una categoría especializada de redes neuronales diseñadas

específicamente para procesar datos con una estructura de cuadrícula, como imágenes. Han demostrado ser extremadamente efectivas para tareas relacionadas con la visión por computador, como la clasificación de imágenes, la detección de objetos y la segmentación semántica.

Una **CNN** típica consta de una secuencia de capas, y cada capa transforma una entrada volumétrica a una salida volumétrica mediante una función diferenciable. Las capas más comunes son: Capas Convolucionales, Capas de Pooling, Capas Fully-Connected y la Capa de Salida.

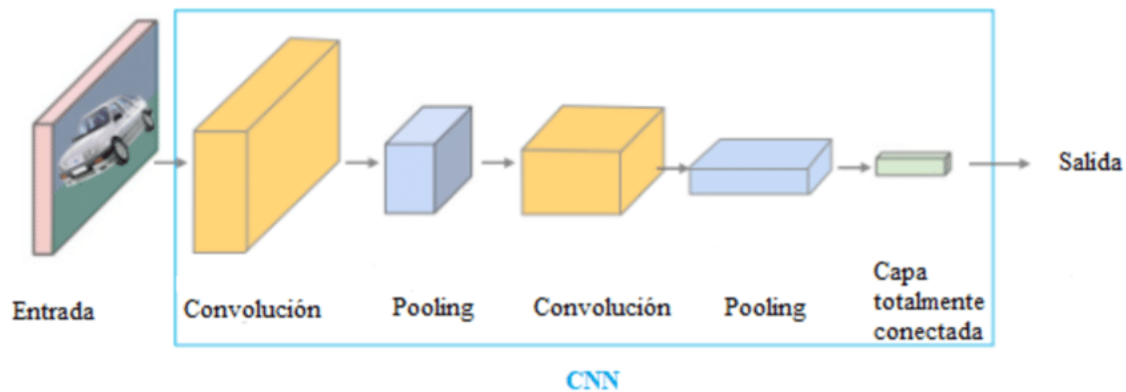


Figura 3: Arquitectura de una red neuronal convolucional [12]

Primeramente, se elige una imagen de entrada, luego se aplican capas de convolución y capas pooling hasta finalmente llegar a una capa totalmente conectada (o “fully-connected” en inglés) seguida de la salida.

Para entender el funcionamiento de la **capa de convolución** se necesita entender las operaciones que transcurren dentro de ella.

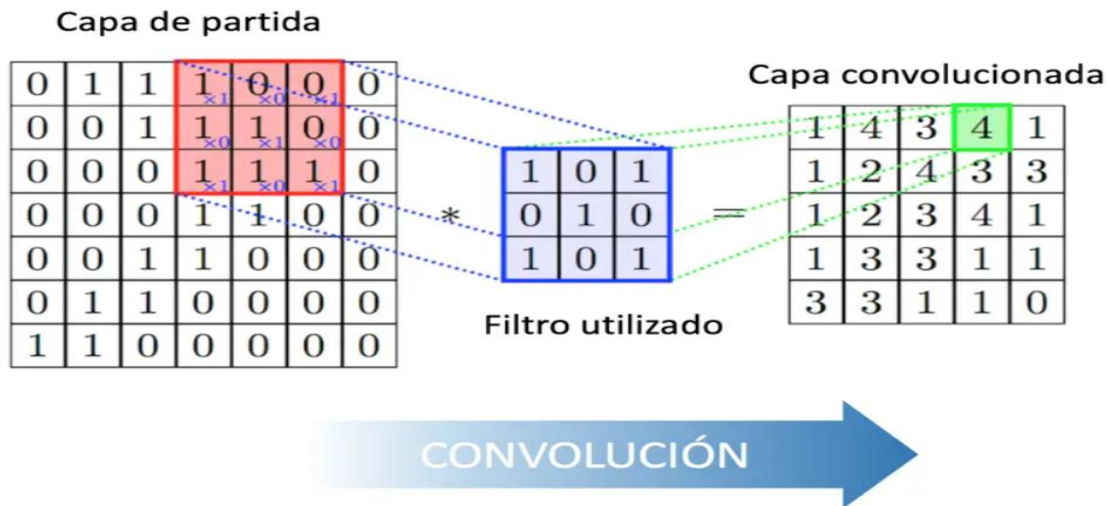


Figura 4: Filtro o “kernel” utilizado en capa convolucional [13]

Como se observa en la *figura 2* utilizamos un filtro o “kernel”. Para transicionar entre la capa de partida y alcanzar la capa convolucionada, nos vamos deslizando en la capa de partida, este deslice depende de un parámetro “**stride**” que nos indica cuantas filas y columnas nos deslizamos en cada iteración. En esta figura se observa que se utiliza “stride” = 1. Luego se efectúa un producto escalar entre la ventana roja observada (que se desliza en las siguientes iteraciones por toda la capa de partida) y el filtro utilizado. El número 4 (veamos la zona verde de la capa convolucionada) es obtenido por ese producto escalar efectuado.

Como se observa la capa convolucionada o “Feature map” es de menor tamaño que la de partida.

La siguiente capa que interviene es la “Capa pooling”, su función principal es reducir el tamaño (ancho x alto) de la imagen, que ayuda a prevenir el sobreajuste. Hay diferentes tipos de pooling, los más comunes son “Max pooling” y “Average pooling”. En “Max pooling” se calcula el valor más alto de la ventana seleccionada y en “Average pooling” se calcula la media de los valores de la ventana.

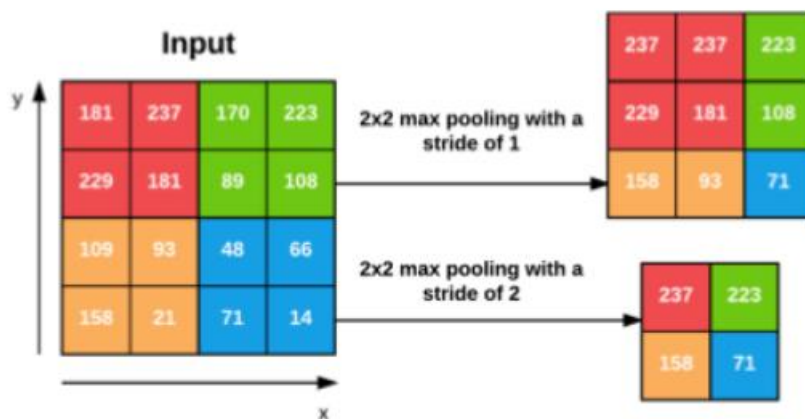


Figura 5: Capa pooling [14]

Después de varias capas convolucionales y de pooling, la alta representación de nivel de la entrada se aplanar y se alimenta a una o más “**capas fully-connected**”. Estas son redes neuronales tradicionales que realizan cálculos en la data, que eventualmente resultan en la salida final, como una clasificación.

La última “**capa de salida**” produce la clasificación final de la entrada. Por ejemplo, en una tarea de clasificación de imágenes, podría indicar las probabilidades de que la imagen pertenezca a una de varias clases.

Aquí se exponen algunas de las características de cómo funcionan las CNN en la Detección de Vehículos:

**Extracción de características:** Al igual que con cualquier otro tipo de imagen, las primeras capas de una CNN diseñada para detectar vehículos identificarán características de bajo nivel, como bordes y texturas. A medida que avanzamos hacia las capas más profundas de la red, estas características se combinan para formar patrones más complejos, como la forma de una rueda, una puerta de automóvil o una ventana.

**Ventanas deslizantes y ROI (Regiones de Interés):** Tradicionalmente, para la detección de objetos, se utilizaba un enfoque de “ventana deslizante”, donde diferentes partes de la imagen se analizaban en busca de objetos (en este caso, vehículos). Sin embargo, con la introducción de arquitecturas como R-CNN (Redes Neuronales Convolucionales Regionales) y sus variantes (Fast R-CNN, Faster R-

CNN), ahora es posible identificar regiones de interés en la imagen que tienen más probabilidades de contener un vehículo, haciendo el proceso mucho más eficiente.

**Regresión de Bounding Boxes:** Una vez que se identifica una región de interés que podría contener un vehículo, la red realiza una regresión para ajustar precisamente un "bounding box" (cuadro delimitador) alrededor del vehículo. Esto asegura que el vehículo sea adecuadamente encuadrado, independientemente de su posición y orientación.

**Clasificación:** Dentro de la detección de vehículos, a menudo es útil no solo identificar la presencia de un vehículo, sino también su tipo (por ejemplo, coche, camión, motocicleta). Las capas finales de la CNN se encargan de esta tarea de clasificación.

**Supresión no máxima (NMS):** En la detección de objetos, es común que se propongan múltiples bounding boxes superpuestos para un único objeto. La supresión no máxima es una técnica post-procesamiento que elimina los bounding boxes redundantes, conservando solo el bounding box con la mayor confianza.

**Detección en tiempo real:** Para aplicaciones en tiempo real, como sistemas de asistencia al conductor o conducción autónoma, se requiere velocidad. Arquitecturas como la usada en el proyecto YOLO (You Only Look Once) y SSD (Single Shot MultiBox Detector) han sido diseñadas específicamente para detectar objetos (incluidos vehículos) en tiempo real, procesando la imagen en una sola pasada y detectando múltiples objetos de diferentes clases.

## 2.4.1 YOLO

En nuestro proyecto hemos utilizado la tecnología YOLO, más específicamente la versión v8 para la detección de objetos. YOLO, que significa "You Only Look Once" (Sólo Miras Una Vez, en español), es un sistema de detección de objetos en tiempo real que utiliza una única red neuronal convolucional (CNN) para detectar y clasificar objetos en una imagen o video. A diferencia de otros sistemas de detección que aplican clasificadores a varias ubicaciones y escalas en una imagen, YOLO examina toda la imagen solo una vez, de ahí su nombre.

Existen varias versiones de YOLO, la más reciente es YOLOv8.

El funcionamiento básico de YOLO empieza (1) dividiendo la imagen de entrada en  $S \times S$  celdas, siendo  $S$  un valor arbitrario, en YOLOv3 se utiliza  $S=13$ . Luego (2) cada celda de la cuadrícula predice un número determinado de bounding boxes (cajas delimitadoras) junto con las confianzas para esas cajas. Estas confianzas reflejan qué tan seguro está el modelo de que la caja contiene un objeto y qué tan preciso cree que es el box. Para cada bounding box, el modelo también predice una distribución de probabilidad de clase sobre todas las clases posibles. En otras palabras, la celda tratará de identificar qué objeto se encuentra en la caja. Por ultimo (3) una vez que se han realizado todas las predicciones, YOLO filtra las bounding boxes basándose en un umbral de confianza y luego utiliza una técnica llamada Non-Max Suppression (NMS) para asegurarse de que solo se conserve el bounding box más preciso para cada objeto detectado[15].

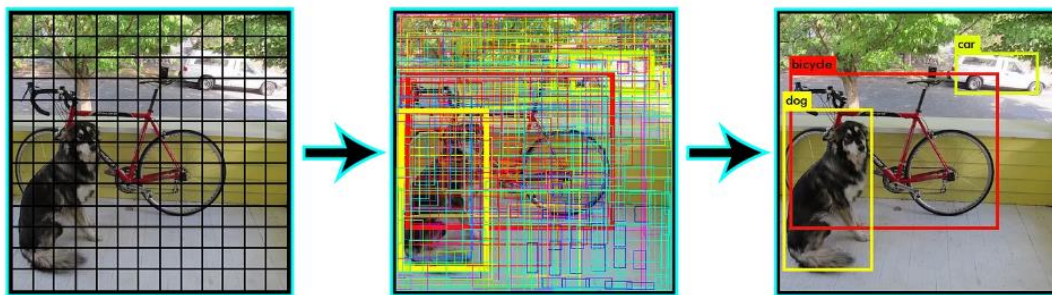


Figura 6: Funcionamiento de YOLO [16]

Como se observa en la figura, la primera imagen representa el paso (1), dividiendo las imágenes en cuadrículas, la segunda en el paso (2) analizando las bounding boxes, y la tercera en el paso (3) concretando las bounding boxes y los objetos que hay en la imagen.

Versiónes más recientes de YOLO, utilizan tres tamaños diferentes de cuadrícula para realizar detecciones en múltiples escalas. Esto permite al modelo detectar objetos grandes, medianos y pequeños.

Algunas de las características que han sido cruciales para la decisión de escoger YOLO como la tecnología del proyecto y que hacen a YOLO una tecnología de alta calidad han sido:

1. **Rapidez:** Una de las características distintivas de YOLO es su rapidez. Procesa imágenes en una sola pasada, lo que lo hace extremadamente rápido en comparación con otros métodos que pueden requerir múltiples pasadas para diferentes tareas. Esta velocidad es crucial para aplicaciones en tiempo real, como la detección de vehículos en carreteras.
2. **Detección en tiempo real:** Debido a su velocidad y eficiencia, YOLO es adecuado para situaciones en tiempo real, lo cual es esencial en aplicaciones de tráfico donde las decisiones deben tomarse en milisegundos.
3. **Precisión:** Las versiones más recientes de YOLO, incluyendo YOLOv4 y, presumiblemente, YOLOv8, han incorporado características avanzadas que mejoran la precisión sin sacrificar la velocidad. Estas características permiten una detección precisa de vehículos de diferentes tamaños y en diferentes condiciones de iluminación y oclusión.
4. **Detección de múltiples objetos:** YOLO no solo identifica la presencia de vehículos, sino que también puede detectar y clasificar múltiples vehículos y otros objetos en la imagen simultáneamente.
5. **Robustez en escenarios variados:** YOLO ha demostrado ser robusto en diferentes escenarios, desde carreteras bien iluminadas hasta condiciones de poca luz o contra-luz, lo que es común en aplicaciones de tráfico real.
6. **Menos falsos positivos:** Las arquitecturas de YOLO tienen menos propensión a los falsos positivos en escenarios de superposición o cuando hay objetos cercanos entre sí, una situación común en el tráfico denso.
7. **Integración con otros sistemas:** YOLO se puede combinar fácilmente con otros sistemas y sensores, como LIDAR o RADAR, para mejorar aún más la precisión y la robustez en sistemas avanzados de asistencia al conductor o vehículos autónomos.
8. **Comunidad activa:** La serie YOLO cuenta con una comunidad activa de desarrolladores y usuarios. Esto significa acceso a actualizaciones, correcciones y mejoras continuas, así como a una amplia gama de recursos para la formación, implementación y optimización de la red.



## 3.Desarrollo

En el presente capítulo, se abordará de forma detallada el proceso de desarrollo del proyecto. El punto de partida es la adquisición de datos, específicamente imágenes y sus correspondientes anotaciones, y su posterior transformación en vídeos. Este proceso garantiza una base sólida para las fases subsiguientes. Una vez consolidada la base de datos, se detallará el proceso de integración y desarrollo en Python con YOLO para detección de vehículos. Seguidamente, se describirá cómo se incorpora la estimación de distancias, el rastreo de vehículos y los desafíos enfrentados en cada etapa. El capítulo también abordará las estrategias de optimización implementadas, fundamentadas en el análisis de resultados previos, y se ofrecerá una descripción de las herramientas y librerías empleadas. Finalmente, se concluirá el capítulo con un resumen y reflexiones sobre lo abordado.

### 3.1 Obtención y Preprocesamiento de datos

Contar con un conjunto de datos robusto y confiable es fundamental para iniciar cualquier proyecto de análisis y modelado. En este proyecto, era esencial disponer de vídeos que no solo sirvieran como insumo para el análisis, sino que también permitieran contrastar nuestras estimaciones con valores reales y verificables.

Ante esta necesidad, se llevó a cabo una búsqueda exhaustiva de fuentes confiables que pudieran proporcionar estos vídeos. Finalmente, se seleccionó el conjunto de datos disponible en [17] por su calidad y pertinencia.

Primeramente, al seleccionar “Download” en la página web se encuentran enlaces a Google Drive:

annotations: [\[google drive\]](#)

lidar data: [\[google drive\]](#)

camera data: [\[google drive\]](#)

Figura 7: Archivos ONCE [10]

**Camera data:** Diferentes carpetas en formato .tar para cada cámara (7 cámaras diferentes en el coche).

**Lidar data:** Una sola carpeta en formato .tar para el LIDAR.

**Annotations:** Una sola carpeta con las anotaciones de cada imagen.

Dentro de cada carpeta se encuentran diferentes carpetas que contienen en las secciones 1 y 2 una agrupación de imágenes .jpg en orden, identificadas por un frame\_id y que juntas forman un vídeo, y en la sección 3 un archivo JSON con las anotaciones de cada imagen.







 000076	272.388.595	272.388.595	Carpeta de archivos
 000080	288.631.465	288.631.465	Carpeta de archivos
 000092	204.977.894	204.977.894	Carpeta de archivos
 000104	201.566.611	201.566.611	Carpeta de archivos
 000113	297.715.719	297.715.719	Carpeta de archivos
 000121	250.596.908	250.596.908	Carpeta de archivos

Figura 8: Carpetas de frames

































	1616343527200.j...	105.446	105.446	Archivo JPG
	1616343527700.j...	107.931	107.931	Archivo JPG
	1616343528200.j...	111.410	111.410	Archivo JPG
	1616343528700.j...	121.014	121.014	Archivo JPG
	1616343529200.j...	130.024	130.024	Archivo JPG
	1616343529700.j...	142.359	142.359	Archivo JPG
	1616343530200.j...	140.419	140.419	Archivo JPG
	1616343530699.j...	129.252	129.252	Archivo JPG
	1616343531200.j...	130.795	130.795	Archivo JPG
	1616343531700.j...	147.223	147.223	Archivo JPG
	1616343532200.j...	115.640	115.640	Archivo JPG
	1616343533700.j...	94.516	94.516	Archivo JPG
	1616343534200.j...	97.622	97.622	Archivo JPG
	1616343534700.j...	100.583	100.583	Archivo JPG
	1616343535199.j...	99.880	99.880	Archivo JPG
	1616343535699.j...	99.074	99.074	Archivo JPG
	1616343536199.j...	96.417	96.417	Archivo JPG
	1616343536699.j...	101.860	101.860	Archivo JPG
	1616343537200.j...	104.692	104.692	Archivo JPG
	1616343537700.j...	98.363	98.363	Archivo JPG
	1616343538200.j...	97.081	97.081	Archivo JPG
	1616343538700.j...	105.033	105.033	Archivo JPG
	1616343539199.j...	103.771	103.771	Archivo JPG
	1616343539699.j...	111.627	111.627	Archivo JPG
	1616343540199.j...	118.195	118.195	Archivo JPG
	1616343540699.j...	128.458	128.458	Archivo JPG
	1616343541200.j...	146.084	146.084	Archivo JPG
	1616343541700.j...	156.858	156.858	Archivo JPG
	1616343542200.j...	169.975	169.975	Archivo JPG
	1616343542700.j...	180.295	180.295	Archivo JPG
	1616343543199.j...	190.466	190.466	Archivo JPG
	1616343543699.j...	166.941	166.941	Archivo JPG

Figura 9: Imágenes JPG

En la siguiente imagen se muestra la posición de cada cámara y LIDAR dentro del coche.

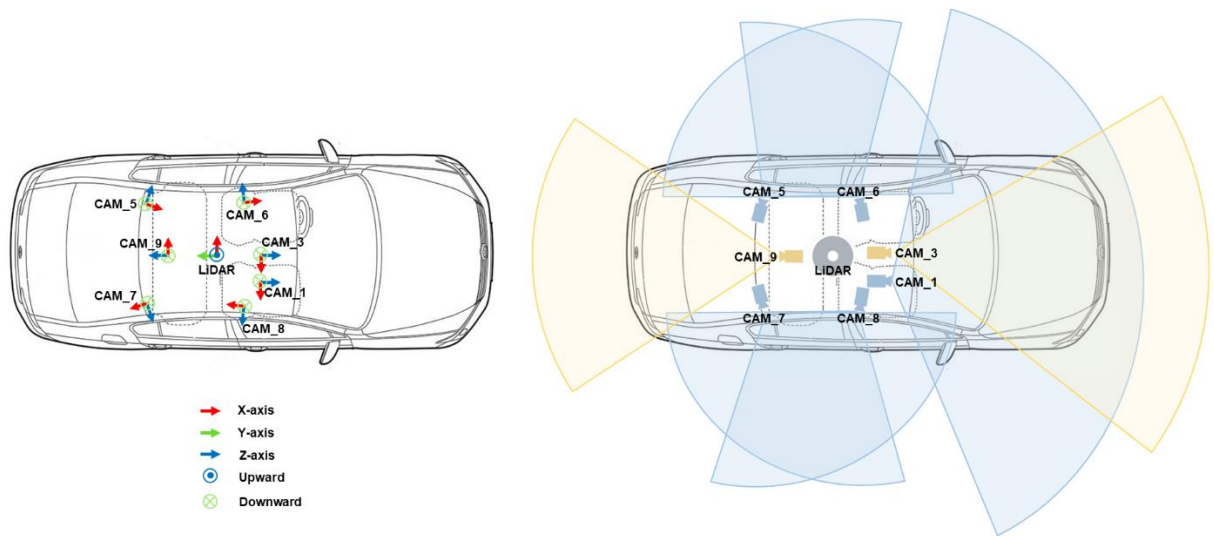


Figura 10: Cámaras instaladas en el vehículo [11]

En el proyecto se ha seleccionado la **CAM\_3** como cámara principal.

En la sección **Annotations** se encuentran los siguientes parámetros fundamentales para detectar los vehículos y conocer su distancia real.

#### meta\_info

*meta\_info* contains basic information about the collecting circumstance and collecting data.

```
"meta_info": {
  "weather":      < str >  -- ["sunny","cloudy","rainy"].
  "period":      < str >  -- ["morning","noon","afternoon","night"].
  "image_size":  < list >  -- (image_width, image_height)
  "point_feature_num": < int > -- features of point cloud.
}
```

#### calib

*calib* contains calibration matrices for all seven cameras.

```
"calib": {
  "cam0[1|3|5|6|7|8]: {
    "cam_to_velo": < list of list > -- 4 x 4 translation matrix from camera coordinate to lidar coordinate.
    "cam_intrinsic": < list of list > -- 3 x 3 intrinsic matrix for camera.
    "distortion": < list > -- 1 x 7 distortion matrix for camera.
  }
}
```

#### frames

*frames* is a list contains information for all frames in each scene, including *sequence\_id*, *frame\_id*, relative *pose* to the first frame of the scene and *annos* for labeled data

```
"frames": [{
  "sequence_id": < str >  -- sequence id of the scene.
  "frame_id": < str >  -- timestamp of the frame.
  "pose": < list >  -- (quat_x, quat_y, quat_z, quat_w, trans_x, trans_y, trans_z).
  "annos": < list >  -- 3D and 2D annotations for each objects appear in this frame, detailed description in frame annotation section.
}]
```

#### frame annotations

*frame annotations* contains the annotation in one scene, including *object name*, *3D box*, *2D box* and *num points in gt*.

```
"annos": [{
  "name": < list >  -- list of object names ['Car','Truck','Bus','Pedestrian','Cyclist'].
  "boxes_3d": < list of list > -- N x 7 bounding box for each object, (cx, cy, cz, l, w, h, θ).
  "boxes_2d": < dict >  -- conatins 2D project bounding box (xmin, ymin, xmax, ymax) for each object in each camera, (-1., -1., -1., -1) otherwise.
  "num_points_in_gt": < list >  -- list of number indicating point cloud numbers in each object 3D bounding box
}]
```

Figura 11: Anotaciones para detectar el vehículo seleccionado [12]

Dentro de los parámetros disponibles, en el proyecto se han utilizado los siguientes:

**'Image\_size'**: Lista de dos parámetros: Ancho y largo de la imagen en pixeles.

**'frame\_id'**: String identificador de la imagen.

**'boxes\_2d'**: Diccionario con 4 parámetros cada elemento correspondientes a los cuatro puntos de la 'Bounding box': 'xmin', 'ymin', 'xmax', 'ymax'. Si el elemento no es detectado en la imagen viene determinado por (-1,-1,-1,-1). *Con esta información podemos detectar el vehículo.*

**'boxes\_3d'**: Lista con 7 parámetros, 3 de ellos seleccionados para el proyecto: 'cx', 'cy', 'cz' correspondientes a la distancia con respecto al centro del vehículo con referencia en nuestro vehículo. *Con esta información podemos detectar la distancia verdadera a la que está el vehículo.*

## 3.2 Detección de vehículos con YOLO

En este apartado se explica como integrar YOLO con Python y que funciones realiza YOLO que nos ayudan en el desarrollo del proyecto.

Primeramente, para utilizar YOLO en Python se debe utilizar la librería "ultralytics" y desde ahí llamar al submódulo YOLO.

```
from ultralytics import YOLO
```

Figura 12: YOLO en Python

Una vez cargado YOLO, se crea una variable **model** en Python que gracias al constructor YOLO y la ruta "yolov8n.pt" se cargan los pesos preentrenados del modelo v8 se utiliza.

```
model = YOLO("yolov8n.pt")
```

Figura 13: Variable YOLO

Al tener **model** como variable disponible en Python se utiliza una de sus funciones más importantes en el proyecto. "track".

Esta función analiza el vídeo que se le indica en los parámetros de entrada, y devuelve como output el vídeo analizado con las bounding boxes de los vehículos

detectados y arriba de cada bounding box un identificador con el número del vehículo (inventado por YOLO arbitrariamente).

*Este no es el vídeo que se muestra en los resultados del proyecto.*

Y una variable Python llamada en el proyecto **results**, con la cual se puede acceder a diferentes parámetros posteriormente explicados.

```
results = model.track(source=video_path, conf=0.3, iou=0.5)
```

Figura 14: Función track

En la función track, se le indican 3 parámetros de entrada.

**source:** El vídeo a analizar.

**conf:** La confianza con la cual se desea que detecte un objeto como válido y se le asigne una bounding box. En este caso es un porcentaje, en el proyecto es 0.3, que indica que si YOLO analiza que un objeto tiene una seguridad mayor del 50% queremos que le asigne una bounding box.

**iou:** Es una métrica que mide la superposición entre dos cajas delimitadoras (bounding boxes). Se utiliza en la etapa de supresión no máxima (NMS, por sus siglas en inglés) para eliminar múltiples detecciones redundantes. Si el IoU de dos bounding boxes es mayor que este umbral, se conserva el bounding box con la puntuación de confianza más alta y se descarta el otro. A establecer iou=0.5, se indica que, si la superposición entre dos cajas delimitadoras es mayor al 50%, una de ellas debe ser eliminada en función de su confianza.

El parámetro **results** contiene una lista en la cual cada elemento es una imagen analizada.

Por lo cual en el proyecto se realiza un bucle “for” para recorrer todas las imágenes y dentro de este bucle se selecciona diferentes anotaciones necesarias.

**‘result.bboxes.xyxy.shape[0]’** : Devuelve el número de objetos detectado en la imagen.

**‘result.bboxes.id’**: Devuelve una lista tensor de IDs detectados en la imagen.

**‘result.bboxes.id[i]’**: Devuelve el ID del objeto i.

**'result.bboxes.xyxy[i]'**: Devuelve la bounding box con los parametros x1,y1,x2,y2 del objeto i.

**'result.bboxes.cls[i]'**: Devuelve un tensor con el número identificador de la etiqueta seleccionada.

**'result.names.get(int(result.bboxes.cls[i]))'**: Devuelve un string con la etiqueta del objeto. En nuestro proyecto el tipo de vehículo.

### **3.3 Estimación de distancia y vehículo seleccionado**

Una vez obtenida la información de todas las bounding box de los vehículos de una imagen junto con su identificador, se calcula su distancia y se realiza un filtro para elegir un único vehículo para analizar.

Como se ha visto en el marco teórico se utilizan las mismas formulas descritas y se añaden como parámetros la anchura y altura del vehículo estimadas a analizar.

Estos son los parámetros dependiendo del tipo de vehículo:

#### **Coche:**

**Anchura:** 1.8metros

**Altura:** 1.5 metros

#### **Furgoneta:**

**Anchura:** 1.9metros

**Altura:** 2 metros

#### **Camión/bus:**

**Anchura:** 2.5 metros

**Altura:** 4 metros

```
def DistanceToObject(finalx1,finalx2,finaly1,finaly2,finalId,count_frames,frame_id_buscado,tipovehiculo):

    NumFramePixelsHeight = 1024
    NumFramePixelsWidth = 1920

    ObjectPixelSizeWidth = finalx2-finalx1
    ObjectPixelSizeHeight = finaly2-finaly1
    # Compute the angle of view of the camera (in radians) in both directions
    AngleOfViewWidth = 120 * (math.pi/180)
    AngleOfViewHeight = 37 * 2 * (math.pi/180)
    # Compute the angle of view associated to a single pixel (in radians) in both directions
    PixelAngleOfViewWidth=AngleOfViewWidth/NumFramePixelsWidth
    PixelAngleOfViewHeight=AngleOfViewHeight/NumFramePixelsHeight
    # Real size of the object in meters
    if(tipoVehiculo == "Coche"):
        RealSizeObjectWidth=1.8
        RealSizeObjectHeight=1.5
    elif(tipoVehiculo == "Furgoneta"):
        RealSizeObjectWidth= 1.9
        RealSizeObjectHeight= 2
    else:
        RealSizeObjectWidth= 2.5
        RealSizeObjectHeight= 4

    # Compute the angle of view associated to the object (in radians) in both directions
    ObjectAngleOfViewWidth=PixelAngleOfViewWidth*ObjectPixelSizeWidth
    ObjectAngleOfViewHeight=PixelAngleOfViewHeight*ObjectPixelSizeHeight
    # Compute the real distance to the object (in meters) according to the apparent sizes in both directions
    DistanceToObject=(RealSizeObjectWidth/2) / (math.tan(ObjectAngleOfViewWidth/2))
    #print(DistanceToObject)
    DistanceToObject2=(RealSizeObjectHeight/2) / (math.tan(ObjectAngleOfViewHeight/2))
```

Figura 15: Estimación de distancia en Python

Se observan dos distancias obtenidas:

**DistanceToObject:** Proporcionada por el parámetro de la anchura del vehículo.

**DistanceToObject2:** Proporcionada por el parámetro de la altura del vehículo.

En el proyecto se ha seleccionado la segunda gracias a la precisión que reporta, también podríamos optar por la primera o realizar una media con las dos.

Una vez obtenida la distancia de cada vehículo de la imagen se selecciona aquel con mayor relevancia a analizar.

Este vehículo seleccionado depende de dos características:

**1-Distancia con respecto a nuestro vehículo:** Se busca un vehículo cuya área bounding box sea mayor que un parámetro específico, en el caso del proyecto, por prueba y error, se establece a **3500** píxeles.

**2-Cercanía del centro del vehículo al centro del eje horizontal:** Una vez ha superado el primer filtro, se busca que el centro del vehículo buscado sea el más



cercano a nuestro centro de referencia, para intentar encontrar un coche que habite en el mismo carril.

### 3.4 Cálculo de diferencia de velocidad

Con el vehículo seleccionado y su distancia se calcula la diferencia de velocidad.

Para ello se calcula el vehículo detectado en una imagen *i* sea el mismo que el detectado en una imagen *j*.

Diferencia de distancia:  $\text{distancia}(j) - \text{distancia}(i)$ .

Diferencia de tiempo:  $\text{tiempo}(j) - \text{tiempo}(i)$ .

Diferencia de velocidad:  $(\text{diferencia de distancia}) / (\text{diferencia de tiempo})$ .

Se expone que el vídeo es grabado a 10 FPS. La diferencia de tiempo entre un frame y el siguiente es de 0.1.

### 3.5 Interfaz gráfica

Para la interfaz gráfica, se ha generado un vídeo a partir de la agrupación de imágenes. Si no se ha detectado ningún vehículo en la imagen, se muestra sin analizar, si se detecta se añaden las diferentes características:

En la parte superior izquierda se observa: Identificador del vehículo, tipo de vehículo, distancia estimada y real, diferencia de velocidad estimada y real.



Figura 16: Muestra de la interfaz gráfica.

En el vídeo se ha destacado con el color verde la bounding box cuando el vehículo seleccionado está más lejos de nosotros que el frame anterior y con color rojo cuando el vehículo está más cerca nuestra.

### 3.6 Optimización y desafíos

Durante el proyecto han surgido algunos desafíos que han ralentizado su ritmo.

**Extracción de anotaciones:** No todos los frames tienen la anotación con la información de las bounding boxes y la posición de los vehículos, para ser más exactos, un frame nos da información y el siguiente no. Esto ha provocado cambios en la función del calculo de diferencia de velocidades.

**Emparejar la bounding box de las anotaciones y la bounding box calculada por YOLO:** Para saber que se refieren al mismo vehículo. Para ello se utiliza la librería **torch** y **torchvision** y en concreto una función llamada `torchvision.ops.box_iou(A,B)`. En la cual devuelve el par de bounding box con el IoU(explicado anteriormente) más elevado.

**Desconocimiento de la unidad métrica de las anotaciones:**

Al principio se empezó a investigar donde podría estar el error, se analizó en una gráfica si la distancia desde centro del vehículo con respecto al centro de la imagen

comparada con la diferencia de distancias reales y estimadas mostraba algún patrón

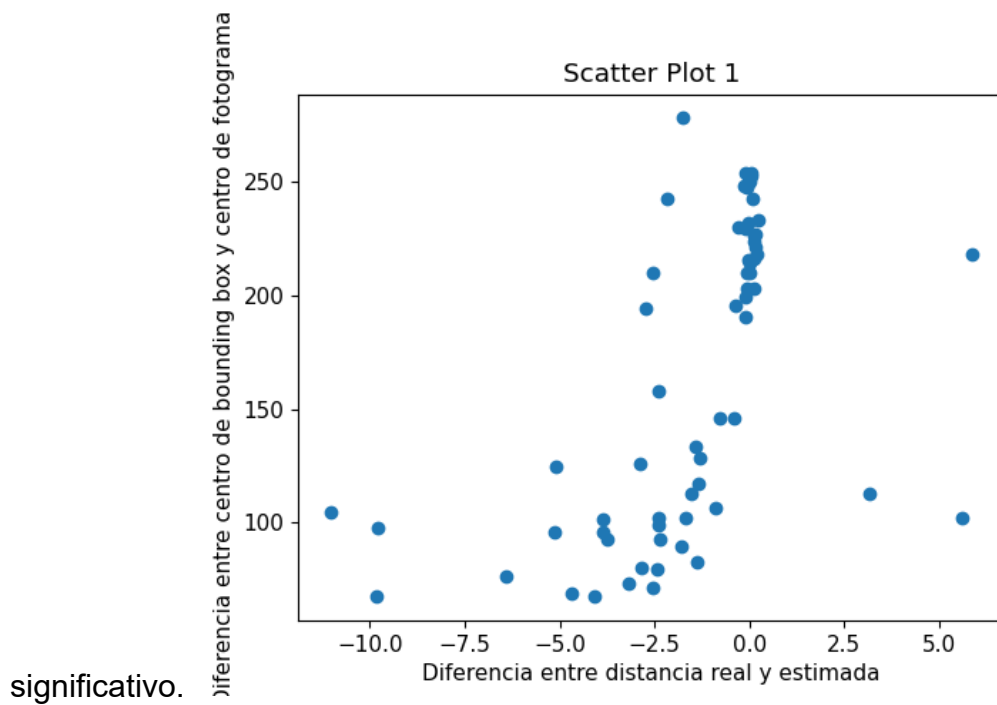


Figura 17: Gráfica de diferencia de centros con respecto a la distancia

Al observarla no se detecta ningún patrón.

Luego se realizó una gráfica donde en el Eje X se muestra la diferencia entre la distancia real y la estimada y en el Eje Y la distancia real.

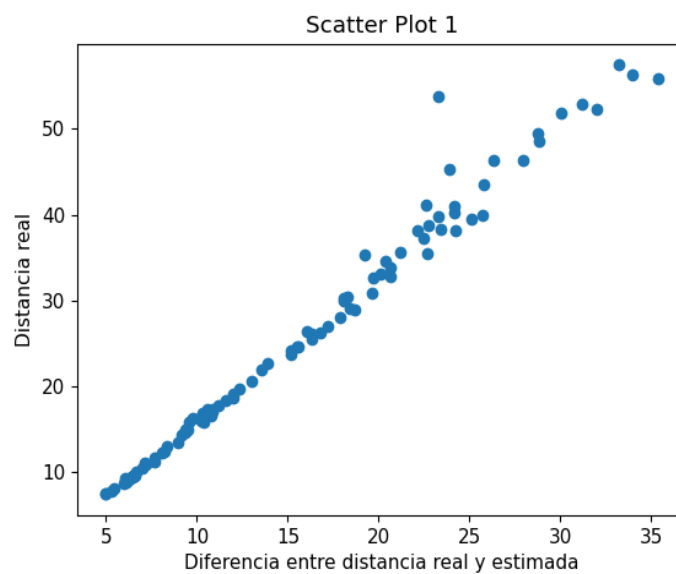


Figura 18: Gráfica para averiguar K

Se observa una línea recta, es decir, existe un factor de conversión **K** que es casi constante para todos los parámetros, este factor se logra al dividir la mediana de las distancias estimadas entre las distancias reales, y da un resultado de **K=0.3739**

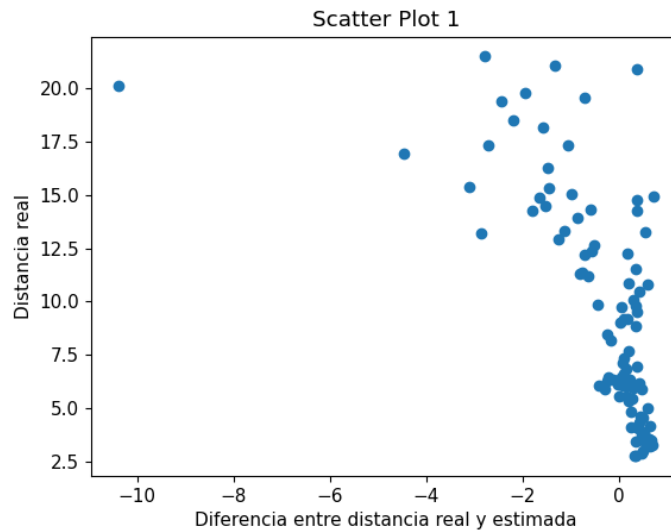


Figura 19: Gráfica con K aplicado

La diferencia entre distancia real y estimada mejora considerablemente, con máximas de 4 metros una vez que supera los 15 metros de distancia.

## 4. Resultados

A continuación, se analizan 4 vídeos formado por 500 frames cada uno, reflejando algunas de las características más importantes a analizar.

### 4.1 Distancia

La distancia a la que se encuentra el vehículo seleccionado repercute en la precisión de nuestro algoritmo, en este apartado vamos a analizar 3 tipos de distancias.

#### 4.1.1 Distancia cercana

Dentro de esta categoría se consideran distancias en un rango de 0 a 5 metros, en la que normalmente el vehículo se encuentra estacionado en algún semáforo u otra situación de parada, y detecta al vehículo que se encuentra delante de él. El algoritmo tiene un error aproximado de 0.5metros, llegando a máximas de 0.8.

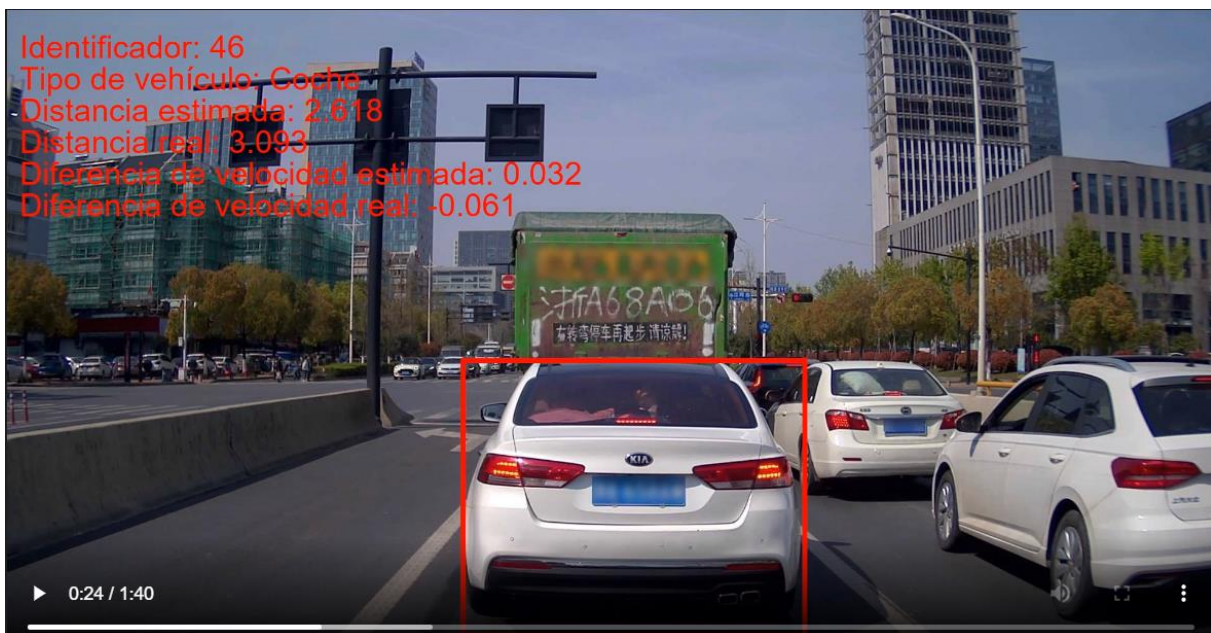


Figura 20: Vehículo en parada 1

La situación de ambos vehículos parados, la diferencia de distancia es de 0.5metros cuando la distancia real es de 3.1metros.



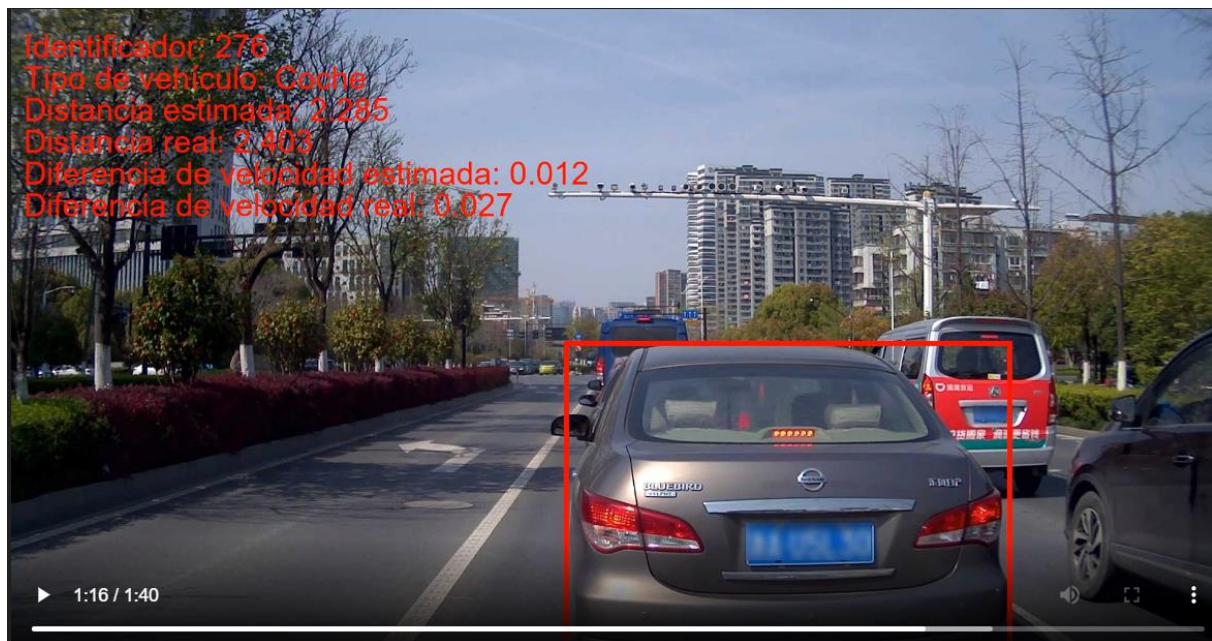


Figura 21: Vehículo en parada 2

Se encuentra la misma situación, pero esta vez el error es inferior a los 0.2 metros.

**IMPORTANTE:** Como se observa en las gráficas hay algunos datos incoherentes. Diferencias de distancias hasta 60 metros como en la gráfica 3, esto ocurre por que YOLO detecta una bounding box que no es la misma de las anotaciones y la detecta como un vehículo. En el análisis se comentarán los resultados cuando las dos bounding box referidas son iguales. Ejemplo:



Figura 22: Error en la detección de bounding box 1



Figura 24: Error en la detección de bounding box 2

Se observan dos figuras, en la primera una detección de un objeto inexistente, y en la segunda se detecta como un “Bus” una señal de tráfico y reporta un error en la distancia de 65metros.

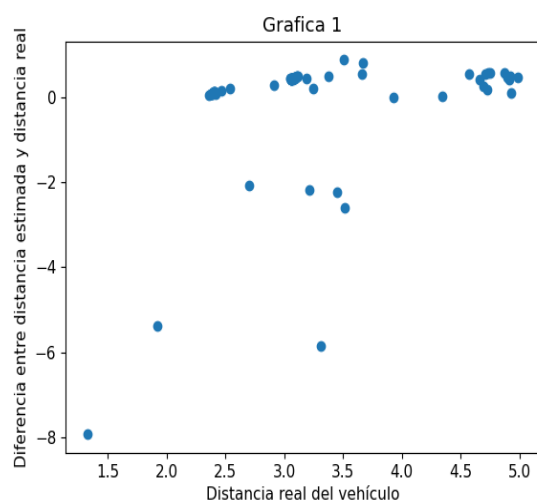


Figura 25: Gráfica distancia cercana 1

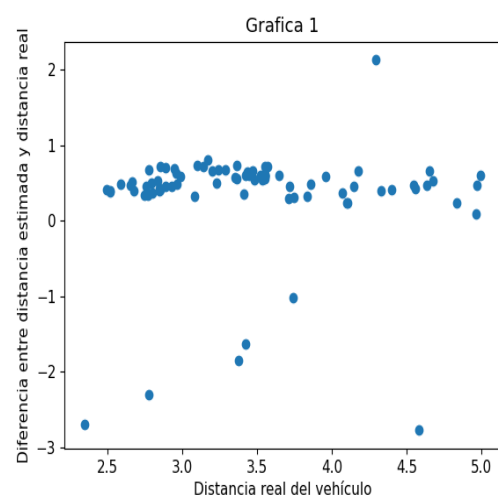


Figura 26: Gráfica distancia cercana 2

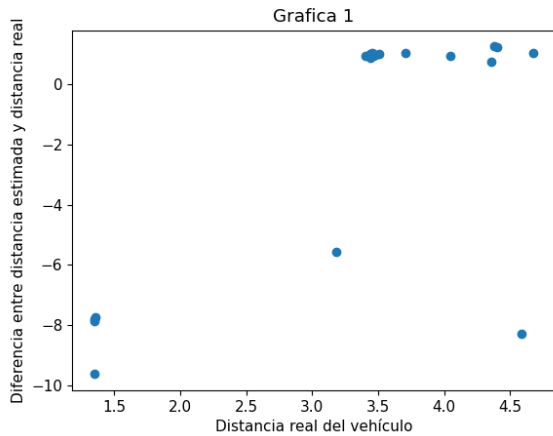


Figura 27: Gráfica distancia cercana 3

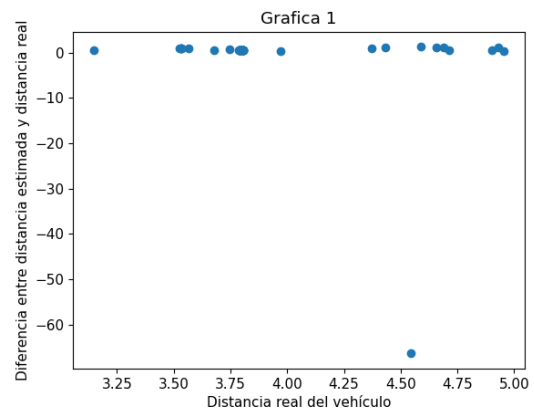


Figura 28: Gráfica distancia cercana 4

### 4.1.2 Distancia mediana

Dentro de esta categoría el vehículo suele encontrarse en movimiento, ya sea por su carril o desviándose.

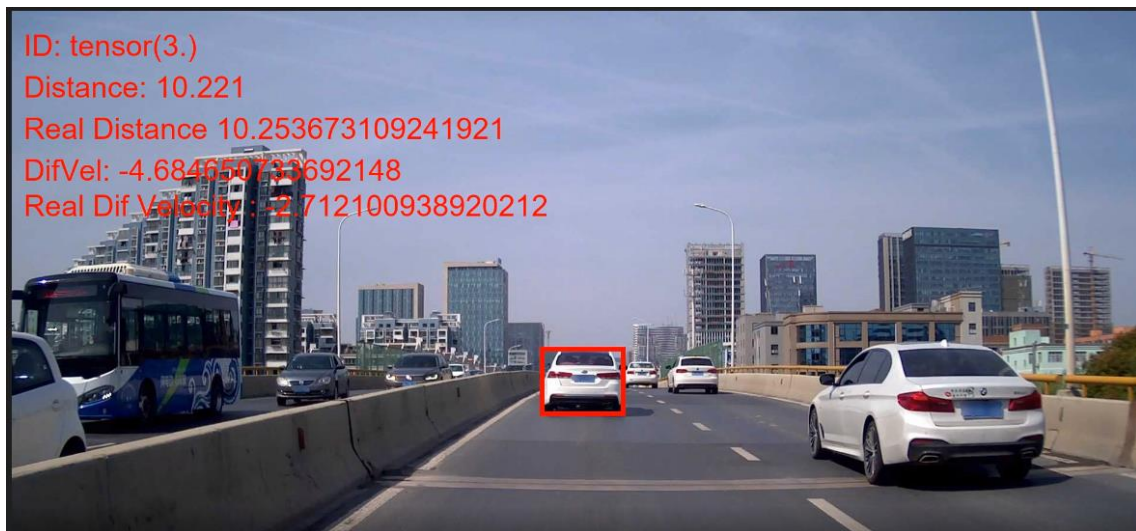


Figura 29: Coche distancia mediana en movimiento

En esta imagen se observa que el error es mínimo y la selección del vehículo es acertada. Con respecto a esta sección se encuentran valores muy cercanos a 0 y los valores más alejados son de 2 metros.



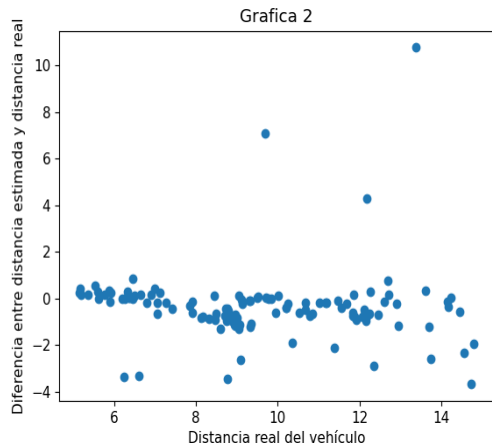


Figura 30: Gráfica distancia mediana 1

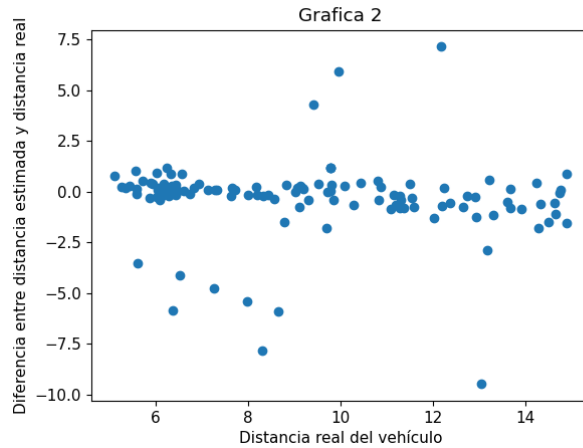


Figura 31: Gráfica distancia mediana 2

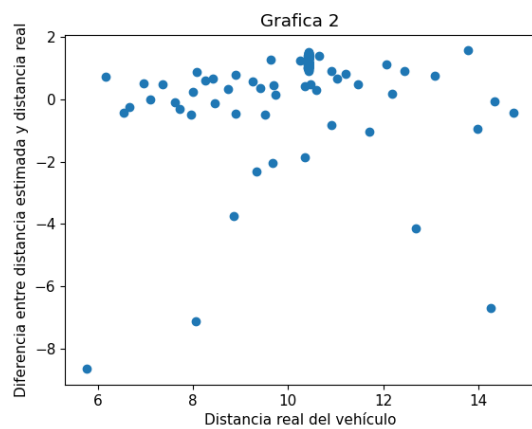


Figura 32: Gráfica distancia mediana 1

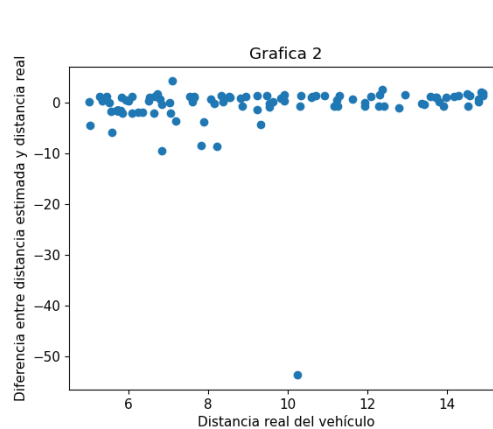


Figura 33: Gráfica distancia mediana 1

### 4.3.3 Distancia lejana

En esta sección se analizan los vehículos detectados a más de 15 metros, usualmente suele ser poco probable que se de esta situación, pero aún así el algoritmo está preparado para ello. El algoritmo suele proporcionar más error, uno por su distancia más lejana y segundo por la falta de visualización entera de la

bounding box, que otros coches la tapan.



Figura 34: Detección vehículo lejano

Se observa el acierto de la detección del vehículo y un error de estimación de unos 1.1 metros.



Figura 35: Detección vehículo lejano obstruido

En esta imagen se observa como los parámetros se modifican por el coche anterior que tapa su visualización.

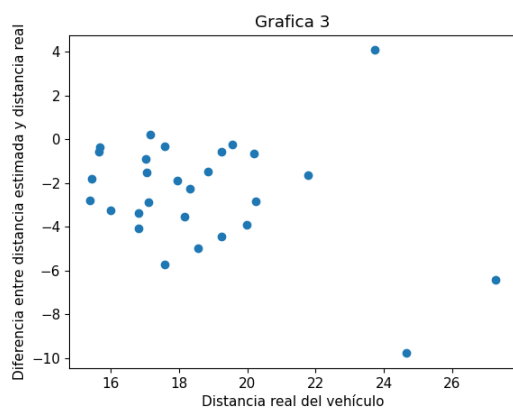


Figura 36: Gráfica distancia lejana 1

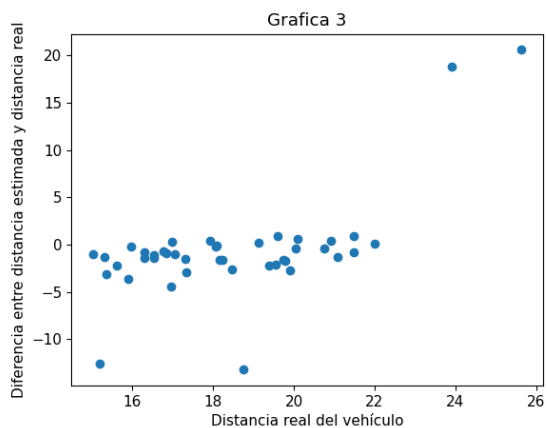


Figura 37: Gráfica distancia lejana 2

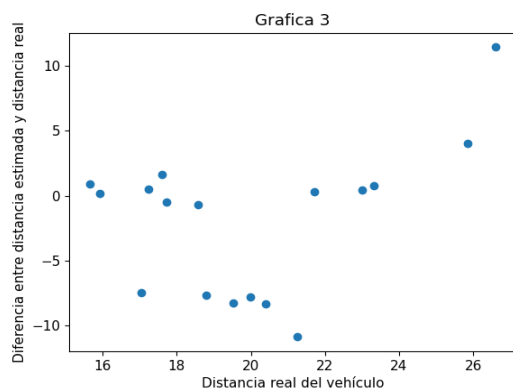


Figura 38: Gráfica distancia lejana 3

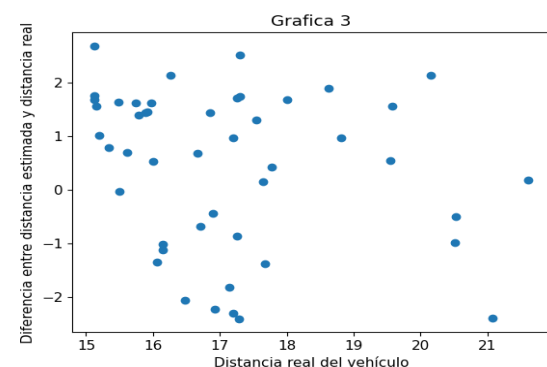


Figura 39: Gráfica distancia lejana 4

A continuación, se muestran las gráficas de distancia sin distinción entre cercana, mediana y lejana.

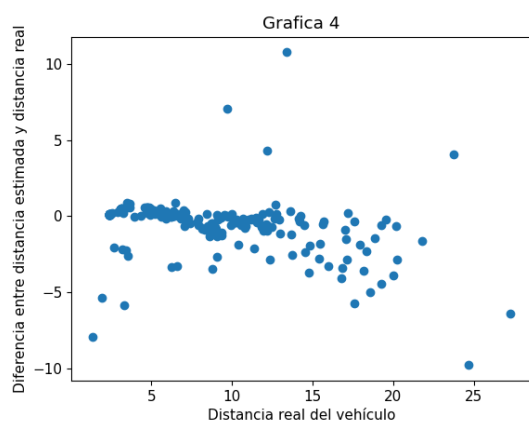


Figura 40: Gráfica distancia 1

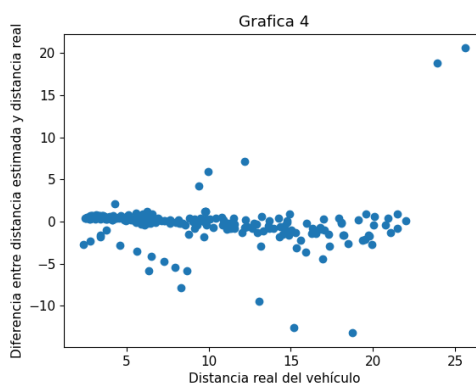


Figura 41: Gráfica distancia 2

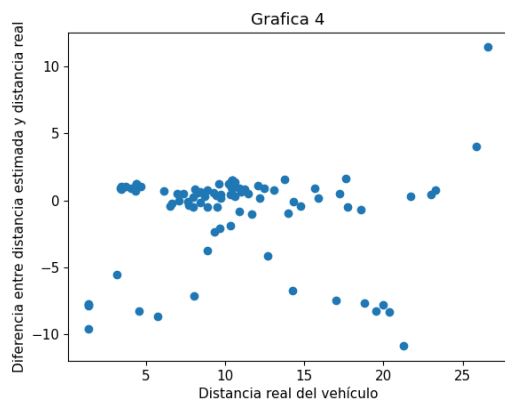


Figura 42: Gráfica distancia 3

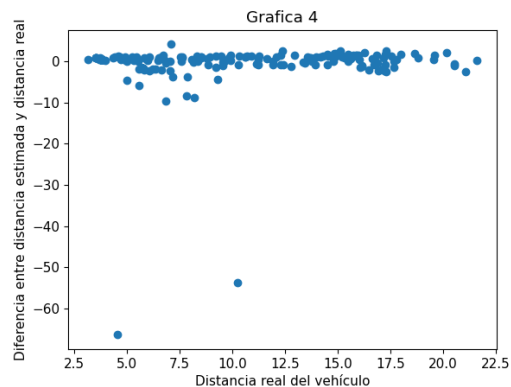


Figura 43: Gráfica distancia 4

### 4.3 Diferencia de velocidad

Se muestran 4 gráficas que miden el error de la diferencia de velocidad, con respecto a la distancia real del vehículo.

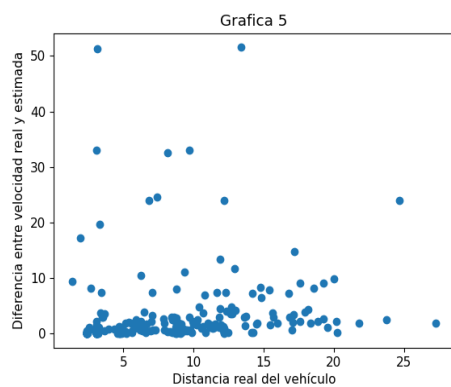


Figura 44: Gráfica dif. Velocidad 1

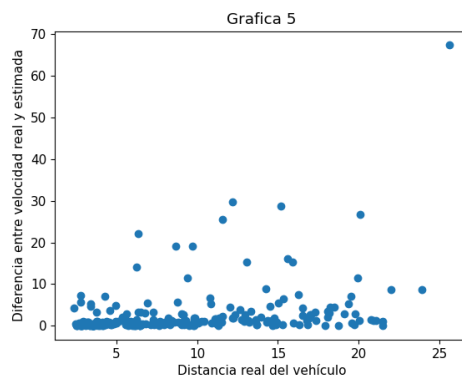


Figura 45: Gráfica dif. Velocidad 2

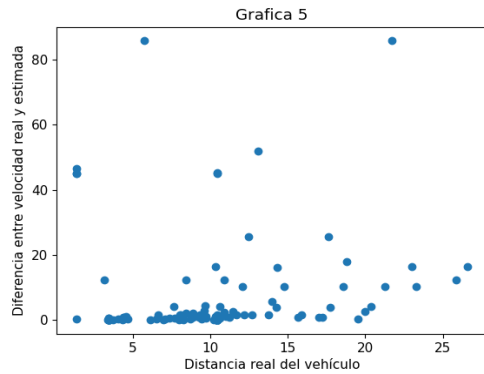


Figura 46: Gráfica dif. Velocidad 3

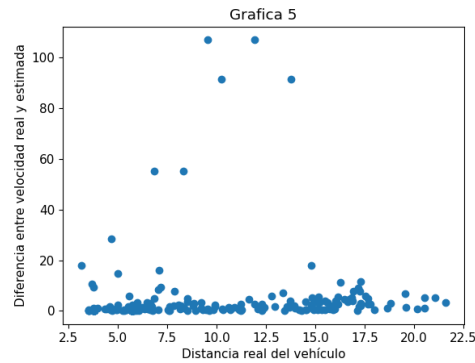


Figura 47: Gráfica dif. Velocidad 4

No se observa un patrón del error con respecto a la distancia real, si no que se mantiene casi constante con respecto a ella.

## 4.3 Tipo de Vehículo

El tipo de vehículo influencia nuestra estimación de velocidad y distancia. En el proyecto se detectan 3 clases de vehículos.

### 4.3.1 Coches

Esta es la categoría con más detecciones y en nuestro proyecto la que mejor resultado proporciona. Esto se debe a que la anchura y altura en los diferentes tipos de coche no se modifica tanto como sucede en furgonetas o camiones, aun así, esa diferencia de dimensiones entre diferentes tipos de coches influencia en nuestro resultado. Aquí se muestran dos figuras y un mapa de calor para mostrar la precisión de nuestra estimación.



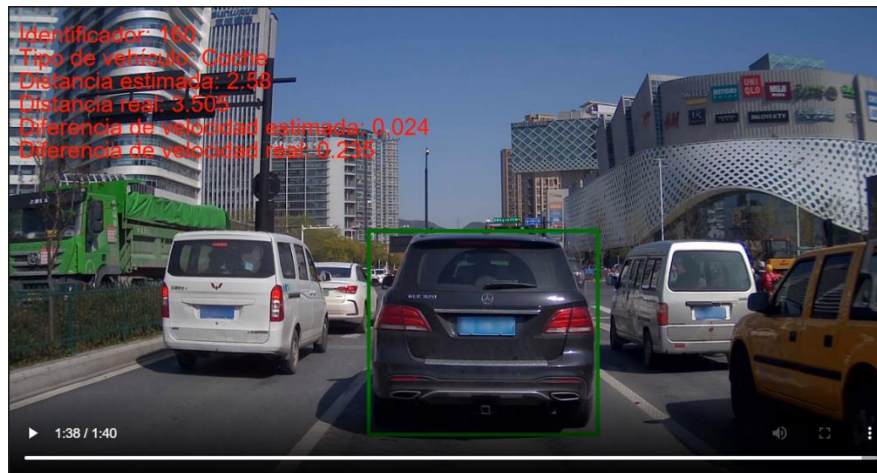


Figura 48: Dimensión del coche



Figura 49: Dimensión del coche 2

Con estas dos figuras se puede observar cómo varía el error con respecto a las dimensiones del coche, en la primera figura se observa un coche más grande, más parecido a una furgoneta y el error es de 1m. En este caso el error es de 0.4 con un coche más parecido al estándar.

### 4.3.2 Furgonetas

Se muestra un histograma con la precisión de la distancia en furgonetas, esta categoría proporciona cierto error por la detección errónea de furgonetas.

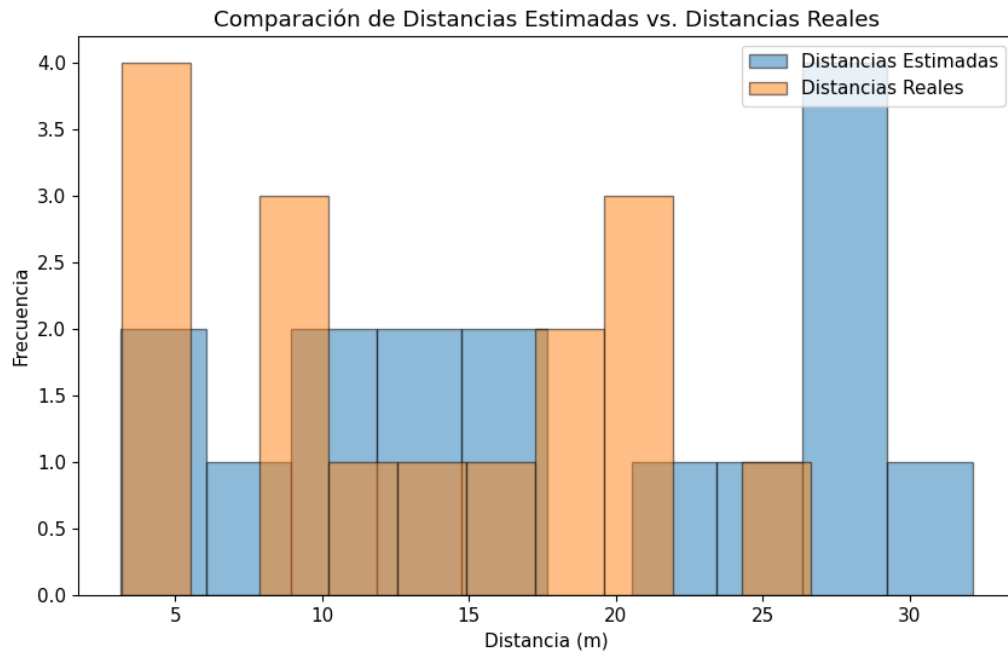


Figura 50: Histograma de precisión Furgoneta

### 4.3.3 Camión

Se ha seleccionado un vídeo especial en el que se detectan dos tipos de camiones en la mayoría de los frames.



Figura 51: Primer tipo camión

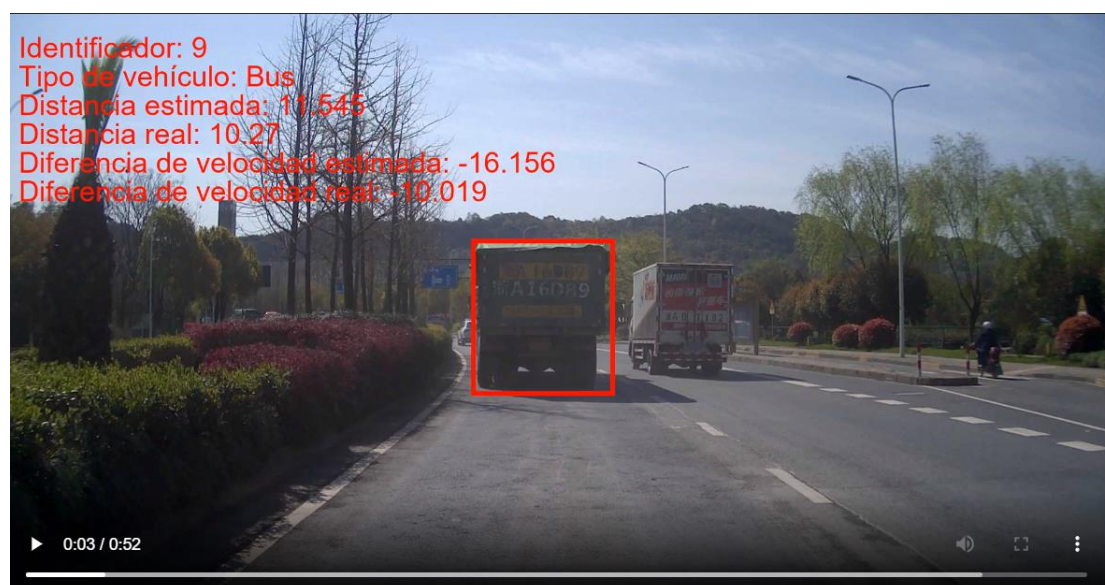


Figura 52: Segundo tipo camión

El vídeo muestra la mayoría de sus frames con el primer camión.



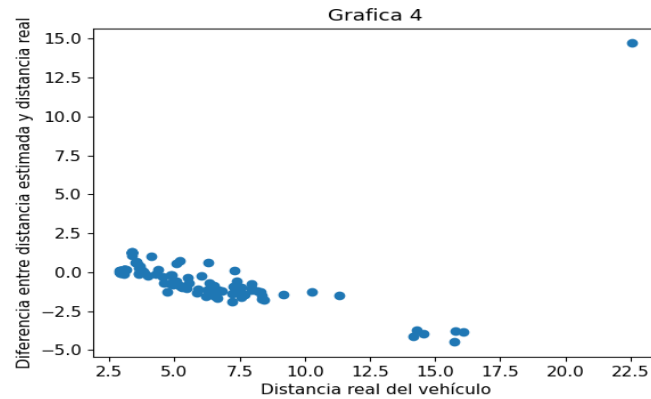


Figura 53: Gráfica de distancia en camiones

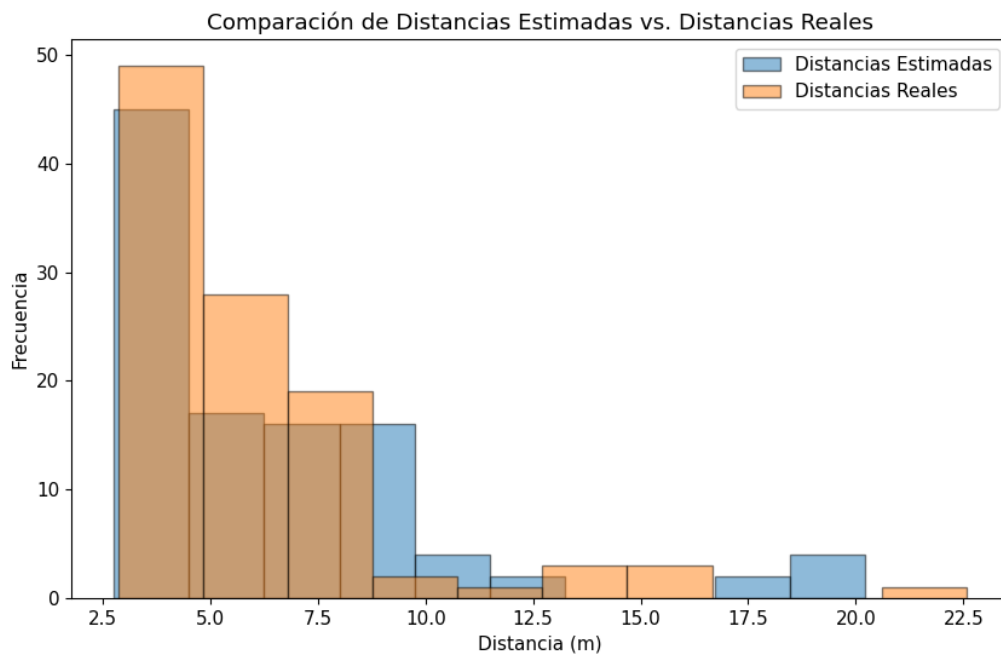


Figura 54: Histograma de precisión en camiones

Los resultados son bastante acertados en la detección de camiones (100 frames de muestra), con errores en torno a 1 metro.

## 4.4 Visibilidad

Factores para tener en cuenta para la detección de vehículos son las condiciones climáticas, la iluminación, deslumbramiento, etc...

Sobre estos factores YOLO detecta los vehículos de manera acertada la mayoría de las veces en condiciones complicadas de iluminación, clima etc.... aunque a veces el algoritmo falla, ejemplos:



Figura 55: Sin detección por cámara borrosa

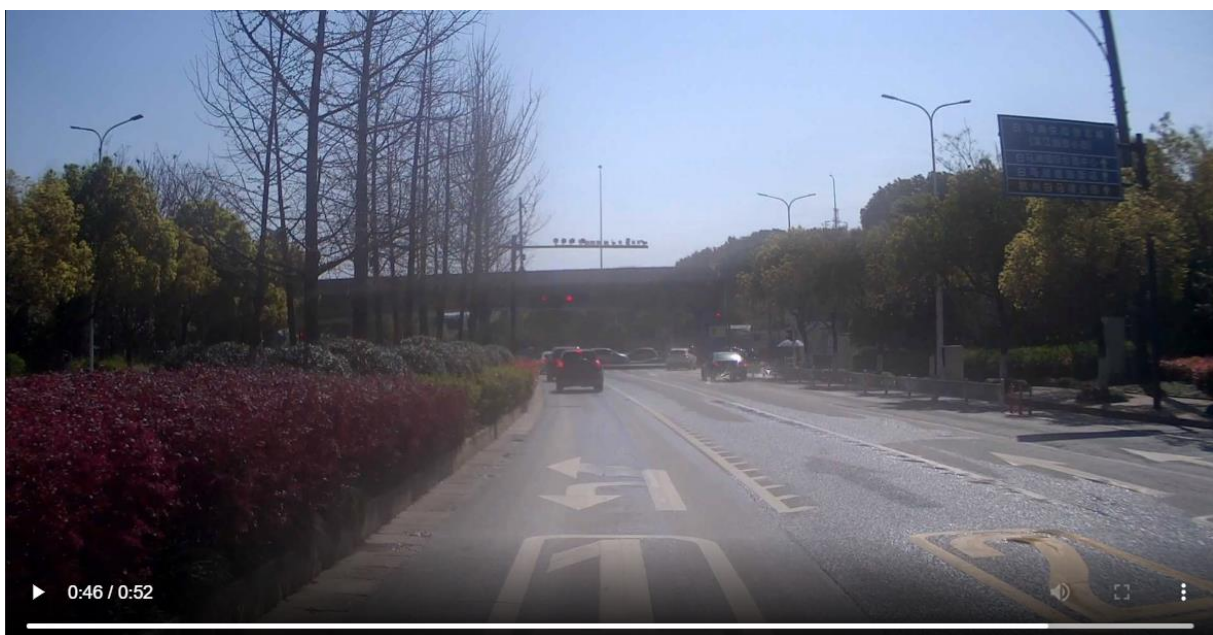


Figura 56: Sin detección por cámara borrosa 2

Si la cámara no está bien enfocada o hay algo de nublosidad puede influenciar en la no detección del vehículo.

## 5.Conclusión

Al cierre de esta investigación exhaustiva, que comprende tanto el análisis empírico de los resultados como la exploración teórica y el desarrollo técnico del proyecto, es posible articular algunas conclusiones significativas.

A través de la implementación del algoritmo YOLO, nuestro sistema ha demostrado una capacidad excepcional para identificar vehículos en carreteras, generando cajas delimitadoras y asignando identificadores únicos. Este componente del algoritmo exhibe un alto grado de eficacia, capturando la vasta mayoría de vehículos y clasificándolos adecuadamente. No obstante, ciertas limitaciones persisten, tales como errores en la delimitación de las cajas o en la asignación de etiquetas. La evolución futura de YOLO promete minimizar estos inconvenientes.

En cuanto a la estimación de distancias, el sistema ha logrado ofrecer aproximaciones notablemente precisas respecto a la ubicación real de los vehículos. Esta métrica podría ser optimizada mediante un análisis más riguroso de las dimensiones efectivas del vehículo, dado que la suposición de dimensiones homogéneas para vehículos dentro de la misma categoría introduce un margen de error considerable. Una solución innovadora podría involucrar la incorporación de un algoritmo complementario que, en conjunto con YOLO, identifique el modelo exacto del vehículo y consulte sus dimensiones oficiales, por ejemplo, en la base de datos del fabricante.

Alternativamente, la adopción de un enfoque de triangulación mediante el uso de dos cámaras podría brindar una mayor precisión en la estimación de la distancia, aunque conlleva una mayor complejidad tanto en la implementación como en el hardware requerido.

De cara al futuro, se podrían desarrollar metodologías para evaluar los riesgos de colisión en carreteras, especialmente si se dispusiera de información sobre la

velocidad del vehículo de referencia. Esto permitiría calcular el tiempo estimado hasta un posible impacto, enriqueciendo así las capacidades predictivas del sistema. Además, se podrían extender estas funcionalidades para prevenir accidentes en entornos urbanos, como la detección y evasión de peatones.

Un avance reciente que subraya el enorme potencial de este ámbito es el caso de un vehículo Tesla que detuvo automáticamente su marcha al identificar a una niña corriendo hacia él.[18]

En resumen, los hallazgos de esta investigación son sumamente prometedores y abren la puerta a un futuro cercano en el que vehículos autónomos no sólo sean una realidad, sino que además contribuyan significativamente a la seguridad vial y a la minimización de accidentes.

## Bibliografía

- [1]: DGT. 1.145 personas fallecieron en siniestros de tráfico durante 2022.  
<https://www.dgt.es/comunicacion/notas-de-prensa/1.145-personas-fallecieron-en-siniestros-de-trafico-durante-2022/#:~:text=En%202022%20se%20produjeron%201.042,y%20previo%20a%20la%20pandemia.>
- [2]: Gonzalez, R.C., Woods, R.E. *"Digital Image Processing"*.
- [3]: Fischler, M. A., & Bolles, R. C. (1981). *"Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography"*.
- [4]: Forsyth, D.A., Ponce. (2002). J. *"Computer Vision: A Modern Approach"*
- [5]: Krizhevsky, A., Sutskever, I., Hinton, G.E. (2012). *"ImageNet Classification with Deep Convolutional Neural Networks"*.
- [6]: *Vehicle Overtaking Hazard Detection over Onboard Cameras Using Deep Convolutional Networks* by Jorge García-González, Iván García-Aguilar, Daniel Medina, Rafael Marcos Luque-Baena<sup>1</sup>, Ezequiel López-Rubio, Enrique Domínguez.
- [7]: <https://yourphotoadvisor.com/does-autofocus-change-focal-length/>.
- [8]: <https://fotoefe.es/angulovision/>
- [9] [10]:  
[https://es.wikipedia.org/wiki/Di%C3%A1metro\\_angular#:~:text=Di%C3%A1metro%20angular%20\(tambi%C3%A9n%20expresado%20a,al%20observador%20en%20su%20v%C3%A9rtice.](https://es.wikipedia.org/wiki/Di%C3%A1metro_angular#:~:text=Di%C3%A1metro%20angular%20(tambi%C3%A9n%20expresado%20a,al%20observador%20en%20su%20v%C3%A9rtice.)
- [12]: [https://www.researchgate.net/figure/Figura-1-Descripcion-del-funcionamiento-de-una-red-neuronal-convolucional-CNN-10\\_fig1\\_348825166/download?\\_tp=eyJjb250ZXh0Ijp7InBhZ2UiOiJfZGlyZWNoIn19](https://www.researchgate.net/figure/Figura-1-Descripcion-del-funcionamiento-de-una-red-neuronal-convolucional-CNN-10_fig1_348825166/download?_tp=eyJjb250ZXh0Ijp7InBhZ2UiOiJfZGlyZWNoIn19)
- [13]: <https://www.diegocalvo.es/wp-content/uploads/2017/07/convoluci%C3%B3n.png>
- [14]: <https://keepcoding.io/blog/capas-pooling-red-neuronal-convolucional/>

[15] [16]: Detección de objetos con YOLO: implementaciones y como usarlas.

<https://medium.com/@enriqueav/detecci%C3%B3n-de-objetos-con-yolo-implementaciones-y-como-usarlas-c73ca2489246>

[17]: [https://once-for-auto-driving.github.io/documentation.html#data\\_annotation](https://once-for-auto-driving.github.io/documentation.html#data_annotation)

[18]: TESLA AUTOPILOT SAVED A LITTLE GIRL.

[https://www.youtube.com/watch?v=bcP6GekID0Y&ab\\_channel=WhamBaamTesla](https://www.youtube.com/watch?v=bcP6GekID0Y&ab_channel=WhamBaamTesla)  
m