Aprendizaje profundo en IoT: Redes neuronales convolucionales con imágenes aplicadas a un vehículo autónomo
Deep Learning in IoT: Convolutional neural networks with Images aplied to an autonomous vehicule

Trabajo de Fin de Máster
Curso 2020–2021

Autor
Alejandro Cordón Ureña

Director
Pedro Antonio González Calero

Máster en Internet de las cosas
Facultad de Informática
Universidad Complutense de Madrid

Aprendizaje profundo en IoT: Redes neuronales convolucionales con imágenes aplicadas a un vehículo autónomo
Deep Learning in IoT: Convolutional neural networks with Images aplied to an autonomous vehicule

Trabajo de Fin de Máster en Internet de las cosas
Departamento de Informática

Autor
Alejandro Cordón Ureña

Director
Pedro Antonio González Calero

Colaborador

Convocatoria: *Septiembre 2021*
Calificación: *8*

Máster en Internet de las cosas
Facultad de Informática
Universidad Complutense de Madrid

24 de SEPTIEMBRE de 2021

Autorización de difusión

El abajo firmante, matriculado en el Máster en Ingeniería Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Aprendizaje profundo en IoT: Redes neuronales convolucionales con imágenes aplicadas a un vehiculo autónomo", realizado durante el curso académico 2021 bajo la dirección de Pedro Antonio González Calero en el Departamento de Informáica, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Alejandro Cordón Ureña

29 de septiembre de 2021

Dedicatoria

A mi familia.

Agradecimientos

A toda mi familia, gracias a quienes soy quien soy y hacia quienes sólo puedo expresar mi sincero agradecimiento por apoyarme durante la etapa académica. A mi director de proyecto Pedro Antonio González Calero por su comprensión en este difícil contexto que nos ha tocado vivir.

Resumen

Aprendizaje profundo en IoT: Redes neuronales convolucionales con imágenes aplicadas a un vehículo autónomo

En el presente trabajo se realiza un estudio, desarrollo y mejora de un vehículo autónomo de bajo coste usando una cámara y aprendizaje profundo. El vehículo aprenderá de un método de conducción autónoma usando OpenCV. Al mismo tiempo se le hace reaccionar a diferentes señales y eventos que pueda encontrar a su paso.

El vehículo se ha montado sobre un proyecto de un vehículo de la marca Sunfounder que usaba sensores de ultra sonidos y detectores de luz inicialmente. Este proyecto ha requerido considerables modificaciones hardware y finalmente no se necesita usarlos.

Para ello se parte de la base de una solución que David Tian realizó con otro hardware, adaptándolo a este vehículo y realizando las actualizaciones tanto hardware como software pertinentes.

Todo el procesamiento de la red neuronal se realiza en el vehículo usando para ello inferencia en una TPU Coral, y una Raspberry Pi 4a para la computación base.

El trabajo se divide en cinco bloques principales. Primero se exponen todas las conclusiones sobre el hardware final y su disposición. A continuación se proporciona el software y la configuración necesaria para el correcto funcionamiento del vehículo. Para continuar se estudia y se adapta la solución de conducción autónoma por OpenCV. En el siguiente punto se trata la conducción autónoma con aprendizaje profundo. Finalmente se añade la detección de objetos y las reacciones del vehículo ante ellos.

Palabras clave

vehículo autónomo, deep learning, edge coral tpu, artificial intelligence, convolutional neural networks, computer vision, raspberry-pi, tensorflow, opencv, picar, python

Abstract

Deep Learning in IoT: Convolutional neural networks with Images aplied to an autonomous vehicule

The present work makes a study, development and improvement of a low-cost autonomous vehicle using a camera and deep learning. The vehicle will learn an autonomous driving method using OpenCV. At the same time it is made to react to different signals and events that it may encounter in its path.

The vehicle has been built on a project of a Sunfounder brand vehicle that used ultra sound sensors and light detectors initially. This project has required considerable hardware modifications and finally does not need to use that sensors anymore.

It is based on a solution that David Tian made with other hardware, adapting it to this vehicle and making the relevant hardware and software updates.

All neural network processing is done in the vehicle using inference on a Coral TPU, and a Raspberry Pi 4a for base computing.

The work is divided into five main blocks. First all the conclusions about the final hardware. The software and settings necessary for the correct operation of the vehicle are provided below. To continue, the autonomous driving solution by OpenCV is studied and adapted. The next point is about autonomous driving with deep learning. Finally, the detection of objects and the reactions of the vehicle to them are added.

Keywords

autonomous-vehicle, deep-learning, edge coral tpu, artificial intelligence, convolutional-neural-networks, computer vision, raspberry-pi, tensorflow, opencv, picar, python

Índice

Índice de figuras

Capítulo 1

Introducción

"En algún lugar, algo increíble está esperando ser conocido"
— Carl Sagan

Actualmente todos los fabricantes de coches se encuentran en una batalla por la seguridad y por una mejor autonomía de sus vehículos. Cada uno permite adquirir diferentes paquetes de conducción autónoma. En función del grado de autonomía cada paquete permite diferentes funciones. Esto ha despertado la curiosidad para querer entender cómo lo hacen.

Para ello, vamos a entender primero qué se entiende por conducción autónoma y que supondría realizar nuestro estudio.

¿Que es la conducción autónoma? Según la SAE Society of Automotive Engenieers hay diferentes tipos de niveles a la hora de clasificar un vehículo autónomo como se puede ver la figura 1.1 y en su página web SAE

Esta escala clasifica las capacidades autónomas de un vehículo en seis niveles diferentes. El nivel cero sería un coche que no presenta ninguna funcionalidad de conducción asistida. Los niveles uno y dos nos describen un tipo de conducción en el que ciertas tareas críticas como acelerar, frenar o manejar la dirección son controladas parcialmente por el sistema, con supervisión humana. En un nivel tres en determinados casos no requeriría de la monitorización continua. En el nivel cuatro en ciertos escenarios limitados, la supervisión humana sería prescindible, por ejemplo carreteras principales con climatología favorable. Si lo logrado por el nivel cuatro fuese posible en cualquier escenario, estaríamos hablando del nivel cinco. El nivel cinco es un nivel de autonomía completa.

Actualmente el mercado de los fabricantes de vehículos está marcado por el desarrollo de la tecnología para mejorar la eficiencia en el consumo o la ayuda en la conducción. Tal es el punto en que existe una escasez en el mundo de los semiconductores que puede afectar directamente a este sector.

En el proyecto que se quiere llevar a cabo entraría dentro del nivel cuatro, ya que en un circuito determinado y con unos elementos determinados pretende ser completamente autónomo.

Actualmente en el mercado se pueden adquirir diferentes paquetes de conducción autónoma. Cada compañía tiene sus nombres. Se ha tomado como referencia la página web de Tesla. Tesla

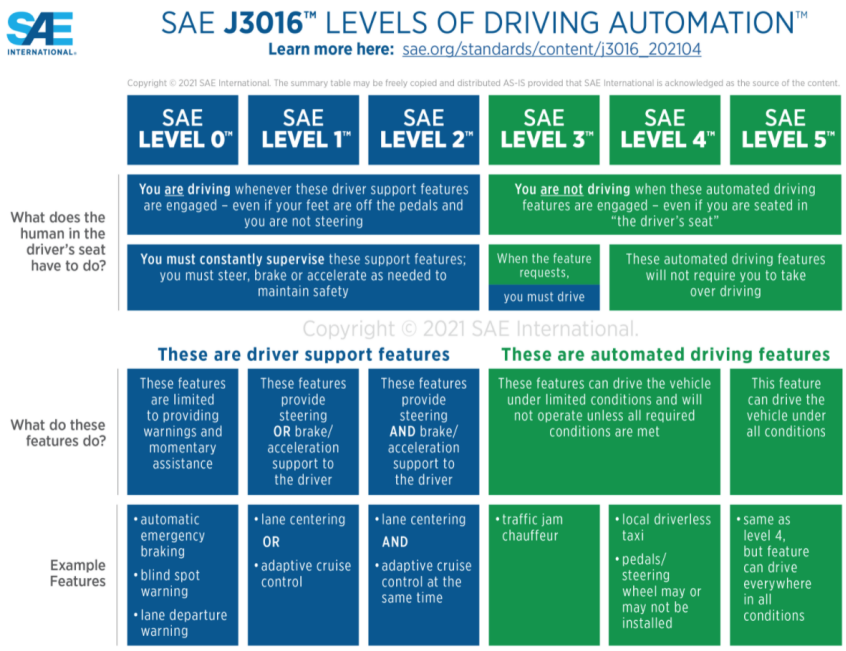


Figura 1.1: Niveles de autonomía según la SAE

En esta página se pueden ver los diferentes tipos de ayuda a la conducción que incluyen sus vehículos.

- Reconocimiento de la velocidad del vehículo que tiene delante
- Ayuda a seguir el carril girando el volante
- Tienen en beta la navegación en Autopilot, que dándole una ruta, te lleva en autopista desde la entrada a la salida de autopista.
- Cambio de carril automático
- Asistencia de aparcamiento
- Permite mover el coche con la aplicación de Tesla
- Reconocimiento de semáforos y señales
- Freno automático de emergencia
- Alarma de colisión frontal y lateral
- Adaptive Cruise Control (ACC)
- Lane Keep Assist System (LKAS)

Estas tecnologías que ayudan a la conducción pretenden detectar y reconocer el entorno que rodea a un vehículo.

Este reconocimiento puede hacerse con múltiples sensores como cámaras, radares, láser o ultrasonidos entre otros.

Éstos sensores permiten formar sistemas mas complejos que puedan permitir montarse de forma modular en cualquier vehículo. Esto supone muchos aspectos técnicos legales y morales. La inteligencia artificial ha sido la llave de todas estas características. Tal es así que todos los fabricantes permiten en mayor o menor a medida la integración de sistemas que ayudan a la conducción de forma inteligente.

1.1. Estudio del arte

Este proyecto es un estudio sobre los vehículos autónomos. Para ello, se ha decidido realizar un vehículo de bajo coste que sirva como entorno de desarrollo. En este entorno se van a integrar todos los proyectos y sensores que permitan llegar a una conducción autónoma lo más amplia posible. También se pretende dejar un entorno abierto para la integración futura de nuevas tecnologías y sensores compatibles.

En este vehículo se tiene la intención de utilizar inteligencia artificial. Para ello se ha integrado una cámara con la que se pretende que el vehículo reconozca su entorno y pueda tomar decisiones respecto a ellas.

También se quieren utilizar todas las herramientas disponibles que puedan acelerar el proceso de integración.

Quizá este proyecto pueda servir de entorno previo para entrenar las tecnologías de los vehículos autónomos del futuro.

Se han encontrado numerosos proyectos de vehículos autónomos en los que podríamos basar nuestro proyecto. Para empezar se requiere encontrar un vehículo base que proporcione unos movimientos básicos y un control del vehículo capaz al menos de acelerar, frenar y girar.

Entre todos los vehículos que se han estudiado, el más sencillo y económico que se ha encontrado es el modelo PiCar-S de Sunfounder. Además la base de procesamiento es una Raspberry Pi que nos proporciona mucho potencial de configuración y posibiliades de actualización.

Estudiando los proyectos que han trabajado en vehículos autónomos, se ha encontrado algunos con los mismos propósitos como Bechtel et al. (2018). Que hace un vehículo autónomo de bajo coste basado en aprendizaje profundo.

Pero hay uno en particular que ha usado un vehículo base parecido. Además comparte una API base de control del vehículo.

Es el proyecto de David Tian DeepPiCar. Este proyecto utiliza un modelo parecido el PiSar-V. Este modelo está dotado de una cámara mecanizada en la parte frontal capaz de moverse para seguir la dirección del carril.

Ambos vehículos comparten un núcleo en la librería que permite girar y configurar la velocidad del vehículo.

El proyecto que queremos hacer es integrar gran parte del proyecto DeepPicar adaptándolo al vehículo PiCar-S.

Además se quiere actualizar la Raspbery Pi por un modelo con mayor capacidad de proceso.

El proyecto DeepPiCar está programado con la versión 2 de Python, versión que se retiró el 1 de Enero de 2020, por lo que se ha compatibilizado el proyecto a la versión 3.

1.2. Motivación

El propósito de este proyecto es crear un vehículo autónomo de bajo coste usando herramientas código libre para aprender el funcionamiento específico de alguno de estos sistemas listados en el apartado anterior.

Ante la complejidad del proyecto se van a usar todos los medios que tengamos disponibles y de código libre y se basará en las aproximaciones que hayan hecho otras personas previamente.

La idea es que el resultado de este proyecto pueda montarse de forma independiente en cualquier vehículo y pueda usarse como entorno de desarrollo para probar futuros sensores o tecnologías que puedan ayudar a la conducción autónoma sin realizar un desembolso económico grande.

1.3. Objetivos

El objetivo de este proyecto es hacer que un vehículo sea capaz de aprender una conducción autónoma.

También pretende ser una base libre y económica para futuros desarrollos donde poder hacer pruebas y desarrollar nuevos sistemas de conducción autónoma.

Este proyecto entra dentro del nivel dos de los niveles propuestos por la SAE, ya que con un tipo de circuito determinado y con unos eventos de tráfico preestablecidos se pretende que el vehículo sea capaz de girar y modificar su velocidad.

La idea es que el vehículo aplique alguno de estos sistemas de conducción autónoma listados anteriormente.

Otro objetivo es utilizar la inteligencia artificial y otros elementos aprendidos en el máster para llevar a cabo estas tareas.

- Implementar un vehículo que sea capaz de circular por un carril de forma autónoma
- Que detecte y reaccione a elementos de su entorno, como peatones y señales de tráfico
- Capaz de aprender nuevos elementos a los que reaccionar

1.4. Plan de trabajo

A continuación se describe el plan de trabajo a seguir para la consecución de los objetivos descritos en el apartado anterior.

- Encontrar una base de vehículo para albergar el proyecto
- Hacer un estudio del arte
- Buscar y estudiar proyectos similares
- Ensamblar e implementar
- Estudiar los resultados

1.5. Organización de la memoria

En esta sección se listan y se hace un pequeño resumen de los capítulos que se van a tratar a lo largo del proyecto.

1.5.1. Hardware

En esta sección se tratan todos los elementos físicos.

El reto que supone montar un vehículo en miniatura de bajo coste que permita montar alguno de los sistemas vistos anteriormente.

Mejoras y optimizaciones fruto del aprendizaje de la realización de este proyecto.

Listaremos y describiremos todos los elementos hardware que necesitaremos para la ejecución del proyecto.

1.5.2. Software

En esta sección se tratan las piezas software que necesitamos.

Se buscarán proyectos de código abierto que nos permitan acelerar el trabajo y adaptarlo a las necesidades del proyecto.

Se mejorará y se integrará cualquier solución que encontremos que nos permita el propósito del proyecto.

Se hará un recorrido para instalar el proyecto desde cero en el Hardware descrito en el apartado anterior.

1.5.3. Proyecto DeepPiCar

Se analizará un proyecto similar que se pueda adaptar a los requisitos de este proyecto.

Se adaptará el proyecto a este vehículo.

1.5.4. Conducción autónoma

En este apartado se adaptará un software para realizar una conducción dentro de los límites del carril.

Esta conducción va a utilizar la cámara únicamente.

Este apartado no utiliza inteligencia artificial.

1.5.5. Entrenamiento y resultados

Se usará el software del apartado anterior para entrenar una inteligencia artificial que aprenda a conducir dentro de los límites de un carril.

1.5.6. Reconocimiento de objetos

Se usará inteligencia artificial para reconocer objetos que puedan modificar el comportamiento del vehículo durante la circulación.

Capítulo 2

Hardware necesario

En este capítulo se muestran todas las adaptaciones y actualizaciones hardware que han sido necesarias para llevar a cabo el vehículo. El objetivo es construir un vehículo de bajo coste capaz de conducir de forma autónoma.

Para ello se ha escogido un vehículo de bajo coste comercial programable, y se ha modificado para que sea capaz de albergar los sensores necesarios para tal propósito.

El mayor reto de este apartado y el que más tiempo ha requerido ha sido el apartado de las fuentes de energía.

A continuación se listan todos los componentes hardware necesarios para empezar.

2.1. Lista de componentes

2.1.1. Sunfounder PiCar-S

El vehículo base que se ha elegido es un Sunfounder Picar-S. Originalmente viene con dos motores motrices traseros y un servo para la dirección delantero. Permite controlar unos sensores de luz para mantenerse en un carril y sensores de ultra sonidos para evitar colisionar con objetos.

Sus baterías originales son dos 18650 que dan una potencia de 2200 mAh cada una a 3.7V, dando como resultado 8.14 Wh.

El vehículo se puede adquirir en su página web: **?** y se pueden consultar los elementos originales.

En la figura 2.1 se puede ver el Sunfounder PiCar-S original.

El fabricante ha publicado una API para poder controlar los diferentes sensores bajo una licencia de software libre. Para ello se conectan varias placas de expansión al puerto GPIO de la Raspberry Pi.

2.1.2. Raspberry pi

Para que el vehículo funcione, debe conectarse y configurarse previamente una Raspberry Pi.

Figura 2.1: Sunfounder PiCar-S

Inicialmente se instaló una Raspberry Pi 3B. Este modelo, aunque funcional, resultó ser lento para este proyecto. Finalmente se decidió que la mejor opción era montar una Raspbery Pi 4B con mayor memoria y mayor capacidad de cálculo. Este modelo también resulta tener un mayor consumo energético.

2.1.3. Tarjeta SD

Se ha usado una tarjeta SD Kingston de 64Gb con una velocidad UHS-I de clase 10 de hasta 100 MB/s

2.1.4. Cámara

Se ha usado una SJCAM SJ4000

Es una cámara para uso deportivo,y aunque se podría usar cualquier cámara usb,se ha optado por esta cámara ya que tiene una batería integrada y puede hacer reducir el consumo energético del conjunto. Es una cámara con una resolución configurable, lo que permite configurar la resolución de los fotogramas de entrada. También tiene una amplitud de lente configurable, lo que resulta muy útil para ver ambos carriles a una distancia corta del suelo.

En la figura 2.2 se puede ver la cámara SJCAM SJ4000.

La cámara se ha acoplado con unos adaptadores al cuerpo del vehículo, quedando en la parte frontal y a una altura suficiente como para ver bien los carriles.

La cámara una vez funcionando debe configurarse en la opción "PC Camera"para que el sistema pueda usarla como una cámara USB.

2.1.5. TPU USB Google Edge

Es un periférico USB compatible con Raspberry Pi que permite inferir modelos de inteligencia artificial. Es una unidad de procesamiento tensorial capaz de ejecutar 4 Tera

Figura 2.2: SJCAM SJ4000

operaciones por segundo y con un consumo energético de 2 Tera operaciones por watio. Es compatible con TensorFlow Lite.

En la figura 2.3 se puede ver el Coral TPU.

TensorFlow models on the Edge TPU - transfer learning on device.

Figura 2.3: Google Edge Coral TPU

2.1.6. Batería

Las baterías que trae el vehículo resultaron ser insuficientes para alimentar la Raspberry Pi 4B, la TPU y los sensores extras que se necesitan para procesar una conducción

(a) Batería Charmast Powerbank 10400mah 15W (b) Batería NetDot 10000 mAh 18W

Figura 2.4: Baterías

autónoma, por lo que finalmente se han montado otras.

Se ha optado por alimentar la Rapsberry Pi 4B y los dispositivos conectados a los USB con una batería batería de 18W - NetDot USB C Power Bank 10000 mAh 18W

La placa de expansión y los sensores se alimentan con otra batería de 15W Charmast Powerbank 10400mah

2.1.7. Elementos de tráfico de juguete

Estas señales de tráfico se usarán en el capítulo de Percepción para que el vehículo reaccione a determinados eventos en el circuito.

2.1.8. Resultado

La cámara debe quedar en la parte frontal y elevada para poder tener una mejor visión general del carril. Para ello se ha acoplado con un adaptador de cámara deportiva a la carcasa del vehículo. Las baterías se han fijado con bridas en la parte inferior del vehículo y la TPU se ha colgado de dos tornillos en la parte superior trasera, lo cual permite una mejor refrigeración de esta.

Conectando todos los sensores y sumándole todas las modificaciones es aspecto final del vehículo es el siguiente.

Figura 2.5: Señales de tráfico

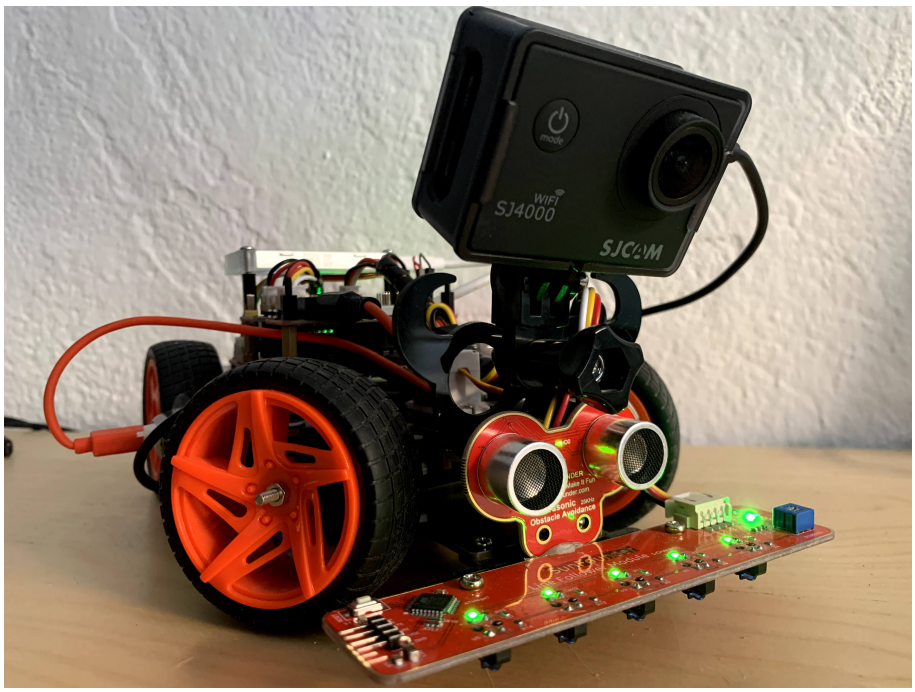


Figura 2.6: Frontal y lateral derecho

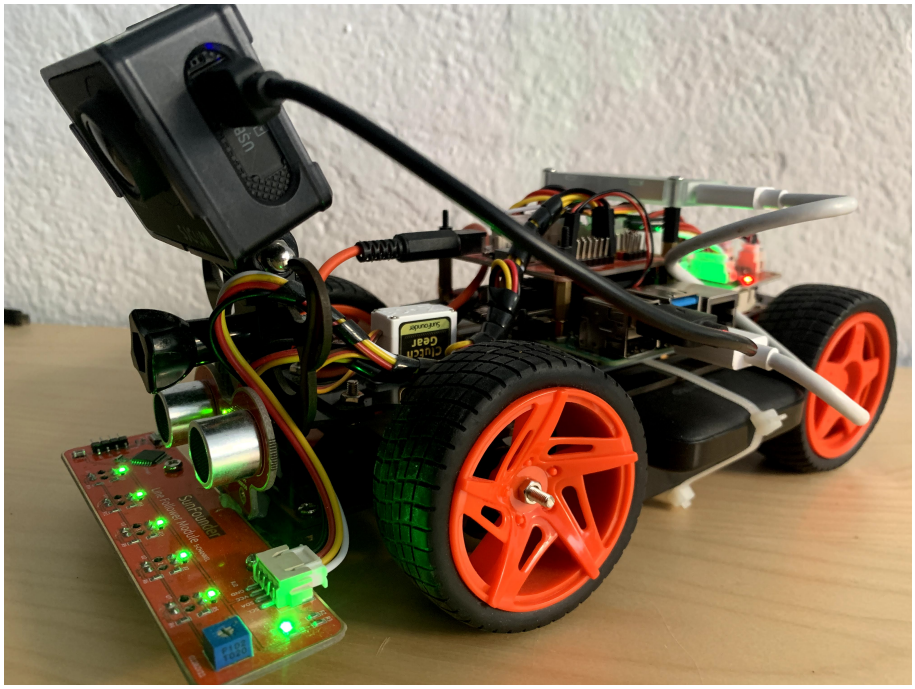


Figura 2.7: Lateral izquierdo

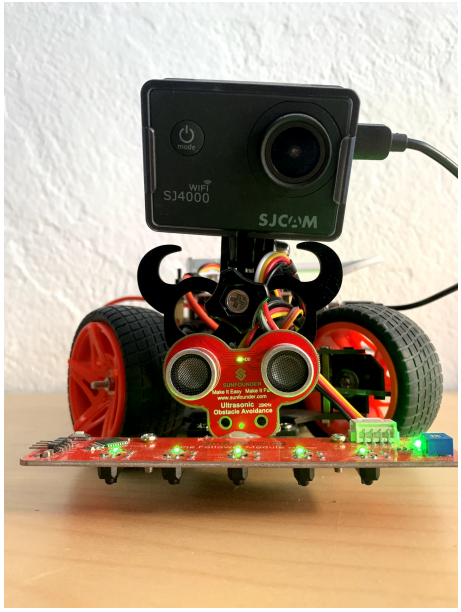


Figura 2.8: Frontal

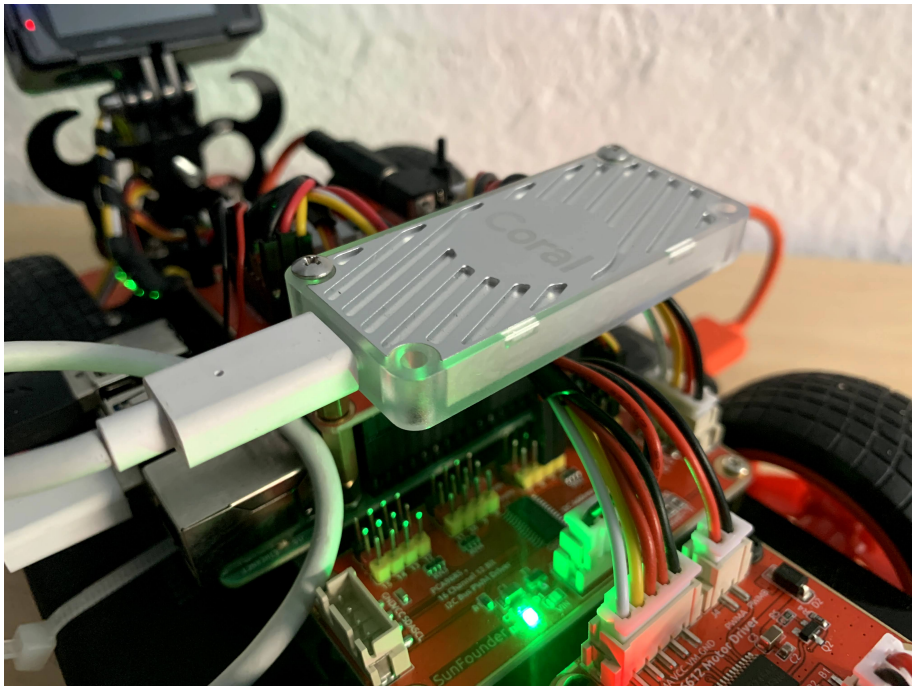


Figura 2.9: Trasera y TPU

Capítulo 3

Software y configuración

Se configurará la visión por computador y software necesario para el aprendizaje profundo. Las herramientas principales que se utilizarán son: El lenguaje Python, La librería OpenCV, que es el paquete de visión por computador, y la librería TensorFlow que es un Framework de aprendizaje profundo de Google.

Todo el software que utilizaremos es código libre.

3.1. Configuración inicial

3.1.1. Instalación de Raspbian en la Raspberry Pi

Se ha optado por la instalación de Raspbian en la Raspberry Pi como sistema operativo base del vehículo. Para ello se ha seguido la guía oficial recomendada por Raspberry.

Figura 3.1: Raspbian

3.1.2. Configuración de acceso remoto

Para acceder de forma remota al vehículo se ha optado por la instalación de SSH y VNC Remote Access.

Se han configurado ambos servicios para que se activen de forma automática al arrancar el sistema.

3.1.2.1. Auto ejecución de acceso remoto al arrancar

Se ha hecho uso del servicio que tiene VNC en la nube para poder acceder de forma remota al vehículo. Debido a que el vehículo no tiene pantalla y no es posible conectar pantallas, el acceso remoto VNC es el más adecuado para ver el estado de la cámara.

3.1.3. Instalación de la cámara

La cámara USB viene instalada en el sistema Raspbian. Para poder visualizar la cámara en el entorno gráfico se han instalado dos aplicaciones Cheese y VokoScreen. Ambos software de escritorio que permiten visualizar la cámara.

3.1.4. Software Sunfounder Picar-S

Sunfounder proporciona una API para manejar el vehículo. Esta API está programada en su gran mayoría en Python y está publicada en Github bajo la licencia: GNU GENERAL PUBLIC LICENSE Version 2

`https://github.com/sunfounder/SunFounder_PiCar-S`

Se ha hecho un fork del proyecto en el que se han integrado todas las modificaciones. El proyecto es TFMIOT y está en Github. `https://github.com/sodapop/TFMIOT`

3.1.5. Configuración OpenCV

Este vehículo se va a centrar la percepción en el vídeo que pueda tomar la cámara. Se va a usar OpenCV para procesar los fotogramas de ese vídeo.

OpenCV es una librería de visión artificial de código abierto que permite capturar, transformar y procesar esos fotograms.

Para instalar OpenCV seguimos la guia publicada en la página web oficial de raspberrypi.

Listing 3.1: Instalación OpenCV

```
pi@raspberrypi:~ $ sudo apt-get install libhdf5-dev -y && sudo apt-get
    install libhdf5-serial-dev -y && sudo apt-get install libatlas-base
    -dev -y && sudo apt-get install libjasper-dev -y && sudo apt-get
    install libqtgui4 -y && sudo apt-get install libqt4-test -y

pi@raspberrypi:~ $ pip3 install opencv-python

pi@raspberrypi:~ $ pip3 install matplotlib
```

Para comprobar que la instalación se ha hecho correctamente, lo más sencillo es importar la librería cv2 en python como se vé en el siguiente código.

Listing 3.2: test OpenCV

```
pi@raspberrypi:~ $ python3 -c "import cv2"
pi@raspberrypi:~ $ python3 -c "import numpy"
pi@raspberrypi:~ $ python3 -c "import matplotlib"
```

3.1.6. Configuracion TensorFlow

Tensorflow es una de las librerías mas usadas en Python para manejar aprendizaje profundo. Esta librería se puede usar de dos maneras. Usando la CPU o un coprocesador tensorial TPU.

3.1.6.1. Para CPU

Este apartado no se va a configurar ya que se va a proceder a usar la unidad de procesamiento tensorial en el siguiente apartado.

3.1.6.2. Para TPU Coral

Para que la inferencia de un modelo de aprendizaje profundo se pueda ejecutar en tiempo real, necesita ser ejecutado en una unidad de procesado tensorial. El coprocesador Coral Edge TPU USB que se ha instalado en el vehículo permite esto.

La compatibilidad de modelos que permite ejecutar este coprocesador tensorial se pueden ver en la siguiente página. Krishnamoorthi (2018)

Para la instalación se ha seguido la página oficial de Coral

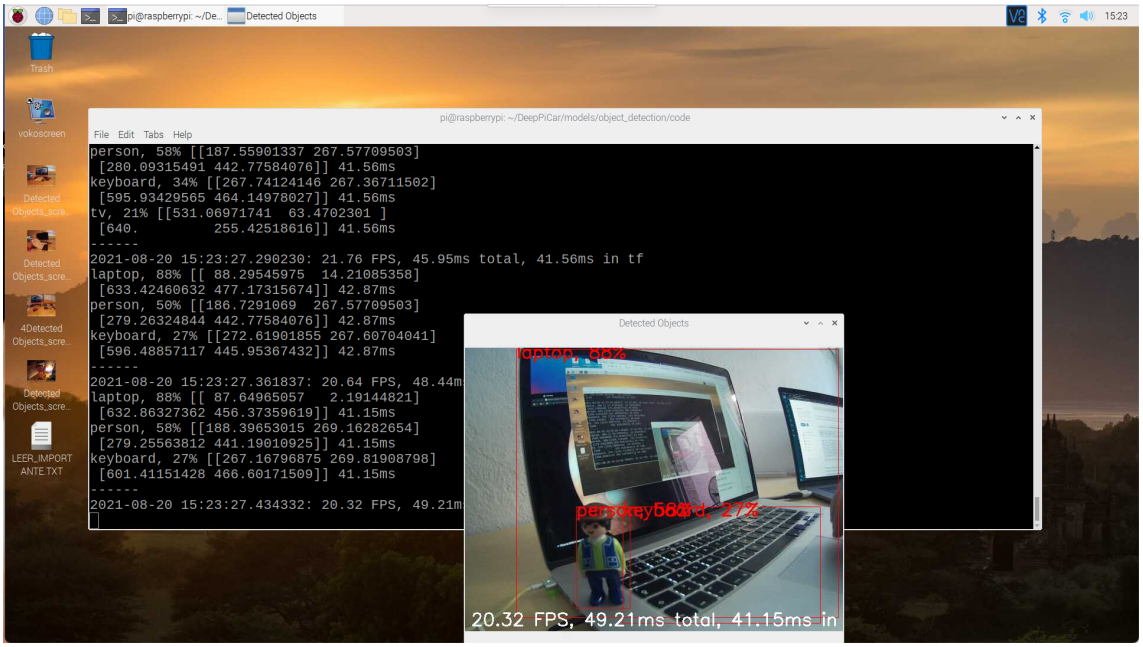


Figura 3.2: Running COCO infereced model

3.1.7. SunFounder PiCar-S Software Configuration

Sunfounder proporciona PiCar API, es un cliente en Python para interactuar con las órdenes basicas del coche.

Esta librería permite establecer la velocidad de las ruedas traseras y gestionar el grado de giro de las ruedas delanteras principalmente.

Tambien facilita el uso de los sensores de ultra sonidos y de los sensores de luz que permiten originamente dejar el vehículo en el carril por el sistema original de Sunfounder.

Información de la distancia de los objetos detectados por el sensor de ultrasonidos

`https://github.com/sunfounder/SunFounder_PiCar-S`

3.2. Proyecto DeepPiCar

Este proyecto es la base y la inspiración inicial de todo el trabajo. Está hecho sobre otro modelo de vehículo y con un hardware diferente. Su autor es David Tian. Tian

Mientras se realizaba el estudio del arte, se vio que hay otros proyectos que realizan lo mismo que se quiere abarcar en este proyecto.

3.2.0.1. Configuración TEST OpenCV

Este proyecto tiene un script para testear la configuración de OpenCV que se va a usar.

3.2.0.2. Configuración TEST COCO Object detection

Este proyecto tiene un script para testear la detección de objetos el código asociado está en object_detection.py

3.3. Proyecto TFMIOT

Este proyecto es un fork del proyecto del Sunfounder para el modelo Picar-S.

Tiene integrado el proyecto DeepPicar y un proyecto hecho durante el máster para el control remoto del vehículo vía bluetooth y MQTT.

He descargado estos dos últimos proyectos y he creado uno nuevo con ambos integrados.

Se he llamado TFMIOT. En él se encuentra todo el software necesario para el funcionamiento del proyecto.

`https://github.com/sodapop/TFMIOT`

- Están los scripts donde he implementado las características siguientes.
- Servidor de control del vehículo por Bluetooth
- Servidor de control del vehículo por MQTT
- Software PiCar-S
- DeepPiCar adaptado a PiCar-S

3.3.1. Licencia

Para establecer la licencia me he basado en las licencias de los proyectos que he usado. DeepPiCar de David Tian tiene GNU GENERAL PUBLIC LICENSE

Capítulo 4

Navegación. Conducción autónoma con OpenCV

4.1. Introducción

En este capítulo se va a usar OpenCV para hacer que el vehículo conduzca de forma autónoma. La idea principal es mantenerlo dentro del carril. Para ello se va a colocar cinta aislante como límites del carril a modo de circuito.

Para poder manter el vehículo dentro del carril se va a detectar el color, los bordes y la dirección del carril. Con esos datos se va a calcular el ángulo de giro de la ruedas directrices para mantener el vehículo dentro del carril.

En este capítulo el vehículo va a ejecutar código explícito que lo mantenga dentro del carril. Es decir, este código se ha codificado para este fin.

En el mundo de la automoción hay dos sistemas que permiten replicar este comportamiento:

- Adaptive Cruise Control (ACC)
- Lane Keep Assist System (LKAS)

ACC es conocido en castellano como control de crucero adaptativo, aunque puede variar su nombre según el fabricante. Es un sistema de control de velocidad que ajusta la velocidad automáticamente al vehículo que llevamos delante con una distancia previamente configurada. Normalmente utiliza como sensores para realizar esos cálculos, un radar, láser o una cámara. Los vehículos equipados con esta tecnología se consideran de nivel 1 de autonomía.

Como trabajo futuro se puede implementar ACC usando el sensor de ultrasonidos integrado.

LKAS es conocido en castellano como un asistente de mantenimiento de carril. Es un sistema para advertir al conductor cuando el vehículo comienza moverse fuera del carril.

Dependiendo del fabricante hay varios tipos:

- Sistema de ayuda a la conducción.

- Sistema de advertencia y corrección.
- Sistema autónomo con ayuda del conductor.

Se va a implementar una imitación de LKAS autónomo usando como base el código de David Tian adaptándolo a nuestro hardware.

Para ello se va a dividir en dos tareas principales:

- Percepción: Detección de los carriles.
- Dirección: movimiento de las ruedas directrices.

4.2. Percepción: detección de los carriles

Esta sección explica cómo se obtiene la dirección de los carriles a partir de un vídeo. Primero se procesa el vídeo para obtener los fotogramas y de cada uno se aísla el color de los límites del carril para obtener una dirección.

4.2.1. Aislar el color del límite de los carriles

Para delimitar los carriles se va a usar cinta aislante de algún color poco común en en el entorno que circule el vehículo.

De cualquier fotograma obtenido por la cámara del vehículo dentro del carril se va a aislar el color de la cinta aislante.

Para hacer esto se va a convertir la imagen de RGB (Red/Green/Blue) a HSV (Hue/-Saturation/Value)

4.2.1.1. Modelo HSV

Intentar aislar el color de los límites del carril de una imagen RGB puede hacer que confundamos los límites según incide la luz en diferentes partes de la cinta.

El color de la cinta puede cambiar desde el punto de vista de la cámara del vehículo según se encuentre con sombras o brillos diferentes en el camino.

El modelo HSV define un modelo de color en términos de sus componentes.

- Hue, en castellano Matiz.
- Saturación
- Valor

Como se puede ver en la imagen el matiz se representa por una región circular. En la región triangular se representa la situación y el valor del color.

El Matiz se representa como un grado de ángulo cuyos valores posibles van de 0 a 360° (aunque para algunas aplicaciones se normalizan del 0 al 100 %). Cada valor corresponde a un color.

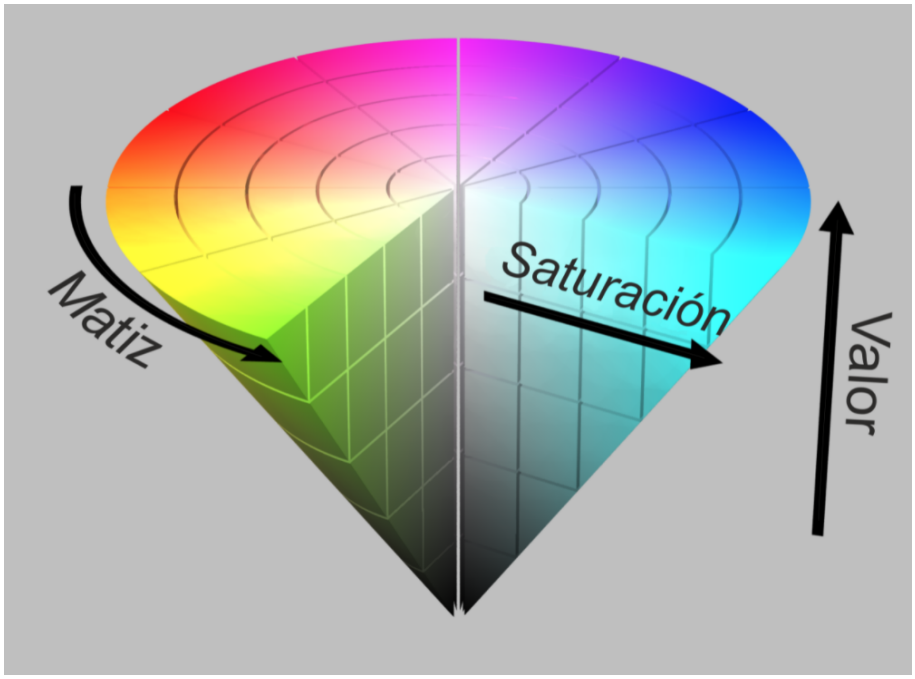


Figura 4.1: Modelo HSV

La saturación se representa como la distancia al eje de brillo negro-blanco. Cuanto menor sea la saturación de un color, más decolorado se verá. Los valores se miden en porcentaje de 0 % a 100 %.

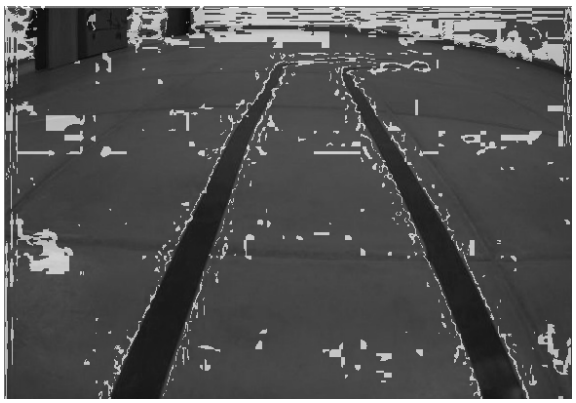
Y el valor representa la altura en el eje blanco-negro. Los valores se miden en porcentaje de 0 % a 100 %. 0 siempre es negro. Dependiendo de las saturación, mas saturado se verá el color.

El componente Hue se dibujara la cinta uniforme al margen de sombras o brillos.

Listing 4.1: RGBtoHSV

```
import cv2
import numpy as np
```

(a) Original



(b) Resultado

```
frame = cv2.imread('test.png')
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

lower_blue = np.array([90, 40, 40])
upper_blue = np.array([150, 255, 255])
mask = cv2.inRange(hsv, lower_blue, upper_blue)

edges = cv2.Canny(mask, 200, 400)
```

Como OpenCV usa por defecto el formato BGR en vez de RGB, se ha usado el parámetro BGRtoHSV en vez de RGBtoHSV. Es una peculiaridad de OpenCV y no afecta al resultado final.

Una vez que la imagen está en HSV se busca remarcar tonos azules en la imagen. Esto se hace especificando un rango de color Azul.

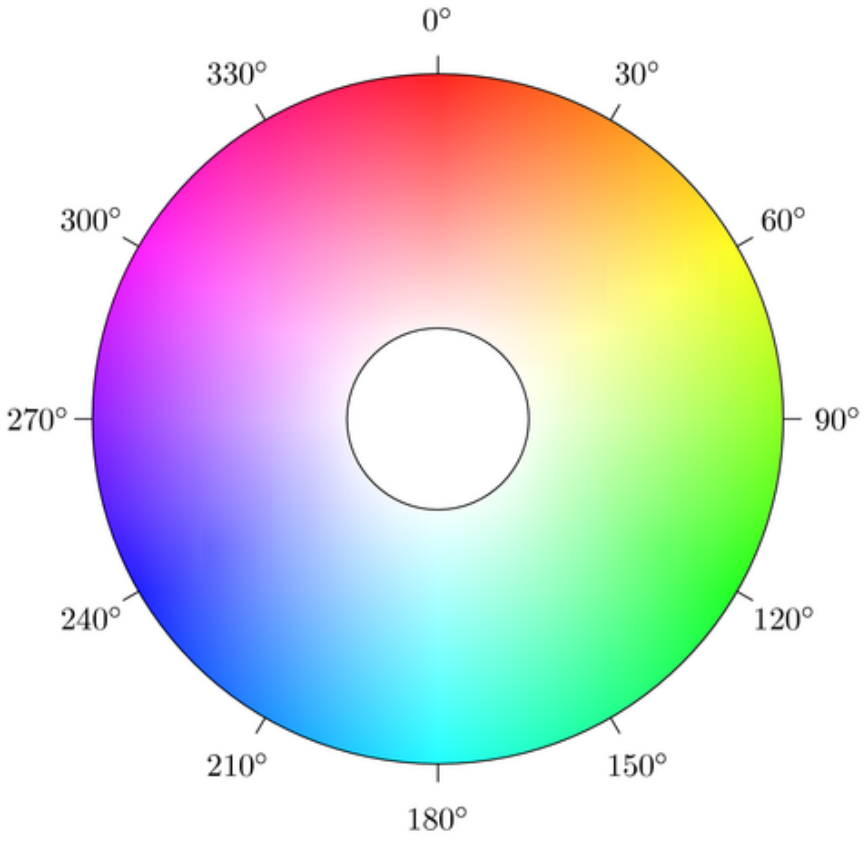


Figura 4.2: Colores HSV

En el siguiente artículo Podpora et al. (2014) se puede leer una explicación más profunda sobre el espacio de color para la interacción hombre-máquina.

Como se puede en ver en la ilustración circular de colores HSV, los valores correspondientes a azul oscilan entre 180 y 300. El color de nuestra cinta está más cercano a 240, pero se va a dejar este rango ya que en el escenario donde va a circular el vehículo, evitamos estos colores.

Estos datos se deben a configurar en función de cada escenario.

OpenCV usa una escala de 180 grados en vez de 360 por lo que el rango equivalente que tenemos que especificar es 90-150 grados.

Los parámetros, Saturación y Valor no son relevantes para la detección del carril.

4.2.2. Detectando bordes de las lineas de los carriles

Para detectar los bordes de la cinta aislante, se va a usar la funcion de OpenCV CannyEdge, que precisamente hace esto.

En el siguiente artículo Intel (2013) se detalla sobre OpenCv y sus métodos.

En la función Canny el primer parámetro es una mascara con los parámetros del punto anterior.

El segundo y tercer parámetro son los valores inferior y superior de la escala de detección.

OpenCV recomienda (100,200) o (200,400), en este caso se usarán este segundo par de valores.

4.2.3. Aislar región de interés

Se pueden detectar otros elementos azules en la parte superior de la imagen. En este caso podemos asegurar que las líneas determinates para la dirección del carril están en la parte inferior.

Por este motivo se va a cortar la parte superior de la imagen y así quedarán las lineas del carril claramente marcadas.

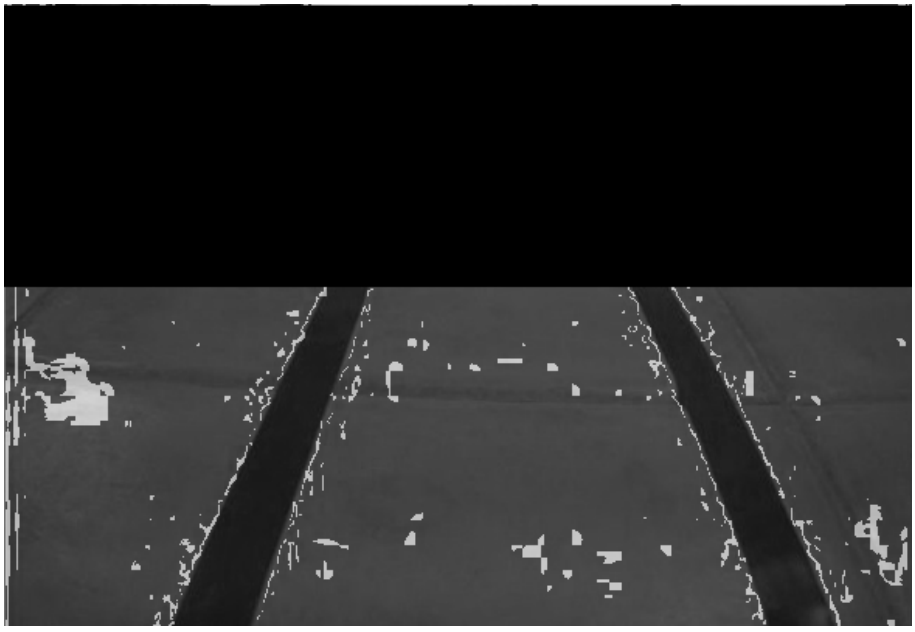


Figura 4.3: Carriles aislados en la parte inferior

4.2.4. Detectar lineas

En de las imágenes con el carril aislado en la parte inferior, se puede ver una serie de píxeles blancos sobre un fondo negro.

Esos píxeles van a determinar las coordenadas de las líneas del carril.

Para hacer esto, OpenCV contiene una función llamada Transformada de Hough que hace exactamente esto.

La Transformada de Hough es una técnica de procesado de imágenes que obtiene figuras como lineas, círculos y elipses a partir de una imagen dada.

En este caso se va a usar para encontrar las líneas de los carriles.

La función HoughLinesP intenta ajustar muchas lineas en los pixels y devuelve la linea que mas se ajusta a ese conjuto de pixels.

En el siguiente artículo se puede obtener más infomración sobre como usar la transformada de Hugh para la detección de figuras Duda y Hart (1975)

Listing 4.2: detect$_{l}ines_{s}egment.py$

```
def detect_line_segments(cropped_edges):
    # tuning min_threshold, minLineLength, maxLineGap is a trial and
        error process by hand
    rho = 1  # distance precision in pixel, i.e. 1 pixel
    angle = np.pi / 180  # angular precision in radian, i.e. 1 degree
    min_threshold = 10  # minimal of votes
    line_segments = cv2.HoughLinesP(cropped_edges, rho, angle,
        min_threshold, np.array([]), minLineLength=8, maxLineGap=4)

    return line_segments
```

Internamente HoughLineP detecta lineas usando coordenadas polares.

HoughLineP tiene los siguientes parámetros:

- Rho: precision de la distancia en pixels

  (usaremos el valor 1)

- Angulo: precision angular en radianes

  (Se usará 1 grado = π / 180).

- min_threshold: número de votos necesarios.

  Para ser considerado linea, un segmento debe tener este mínimo de votos

- min_linelenght: Longitud mínima de un segmento en pixels para que sea considerado segmento.

- MaxLineGap: maximo hueco de separacion entre dos segmentos para ser considerado linea.

Configurar estos parámetros es un proceso de prueba y error.

4.2.5. Límites del carril

La trasformada de Hough devuelve segmentos (x1,y1), (x2,y2). Se van a combinar los segmentos que definen cada linea del carril.

Como se puede observar en la imagen 4.3 en la linea izquierda hay dos lineas con pendiente positiva y en la derecha otras dos con pendiente negativa.

Se puede clasificar cada línea por su pendiente y así obtener solo dos lineas con el límite del carril

Una vez que se han clasificado, la función average_slope_intercept.py calcula la media de las pendientes y la intersección de los segmentos de cada linea.

Obtiene la pendiente de una linea y la intersección, y devuelve el final de la linea.

Además de esa lógica hay dos casos especiales a tener en cuenta:

- Imagen con una sola linea de carril:

  Normalmente se esperan dos líneas pero a veces puede ocurrir que solo se detecte una, bien por un giro pronunciado o por un error. Por eso se hace la comprobación len(right_fit)>0 y len(left_fit)>0

- Lineas verticales

  A veces puede ocurrir que una de las lineas pueda ser completamente vertical. Es un caso que ocurre muy pocas veces y el calculo de su pendiente puede volverse complejo, en ese caso, simplemente las ignoramos. No afecta mucho al resultado final.

4.2.6. Resumen de detección de carriles

Con las siguientes funciones se puede llegar pintar las lineas de los carriles.

- detect_lane - dado un fotograma, devuelve las coordenadas de las lineas de los límites del carril.
- detect_lines_segment - devuelve los segmentos de las lineas
- average_slope_intercept - calcula las pendientes y distigue entre limite izquierdo y derecho
- make_points - ayuda a la función anterior a calcular las pendientes
- display_lines - dibuja las líneas

4.3. Dirección: Giro de las ruedas directrices

Ahora que se conocen los limites del carril identificados, se puede que hacer girar la dirección.El objetivo es mantener el coche lo mas centrado en el carril.

4.3.1. Con dos lineas detectadas

En la situación ideal de identificar dos lineas en la imagen, se usarán para calcular la dirección.

La cámara está situada unos centímetros a la izquierda, pero eso no va a afectar a la hora de determinar la dirección. De hecho en cada fotograma los carriles nacen en el centro de la parte inferior de la imagen, a pesar de esta desviación de la cámara, y la linea de dirección siempre será la bisectriz de las lineas de los limites del carril.

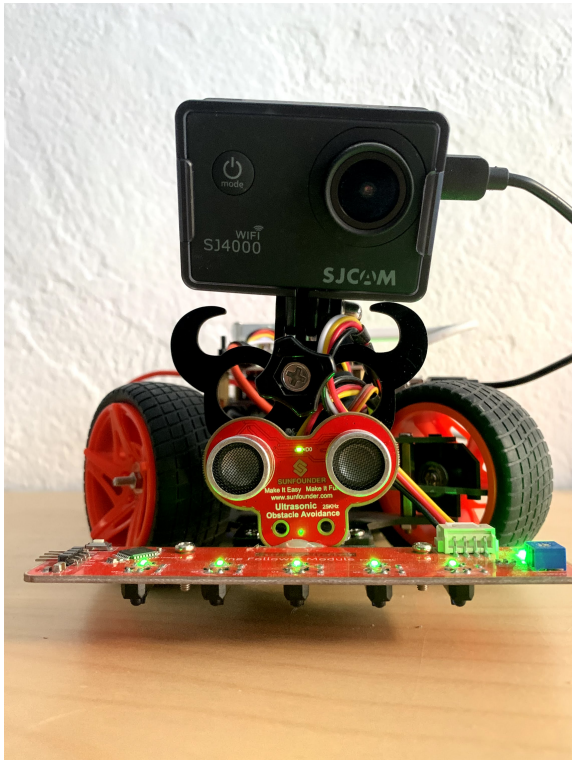


Figura 4.4: Frontal del vehículo

4.3.2. Con una linea detectada

Si solo se detecta una linea de límite de carril, muy probablemente se deba a nos hemos salido un poco del carril o que no esté marcada.

En esta situación, no podemos calcular bisectriz por lo que la dirección es paralela a la que hemos detectado.

redheading_one_line.py

4.3.3. Ángulo de giro

Una vez calculada la linea de dirección, se debe convertir a un dato compatible con servo de dirección

Para esto se tiene que tener en cuenta cómo interpreta la librería PiCar los ángulos de dirección. El código encargado de esto está en coord_to_steearing_angle.py

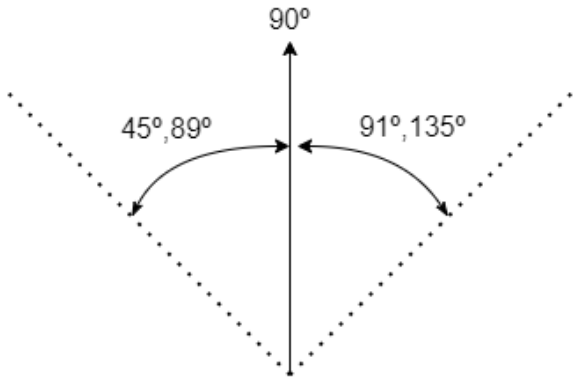


Figura 4.5: Ángulos Picar

4.3.4. Mostrar linea de dirección

Para mostrar la linea de dirección se usará el código display_heading_line.py

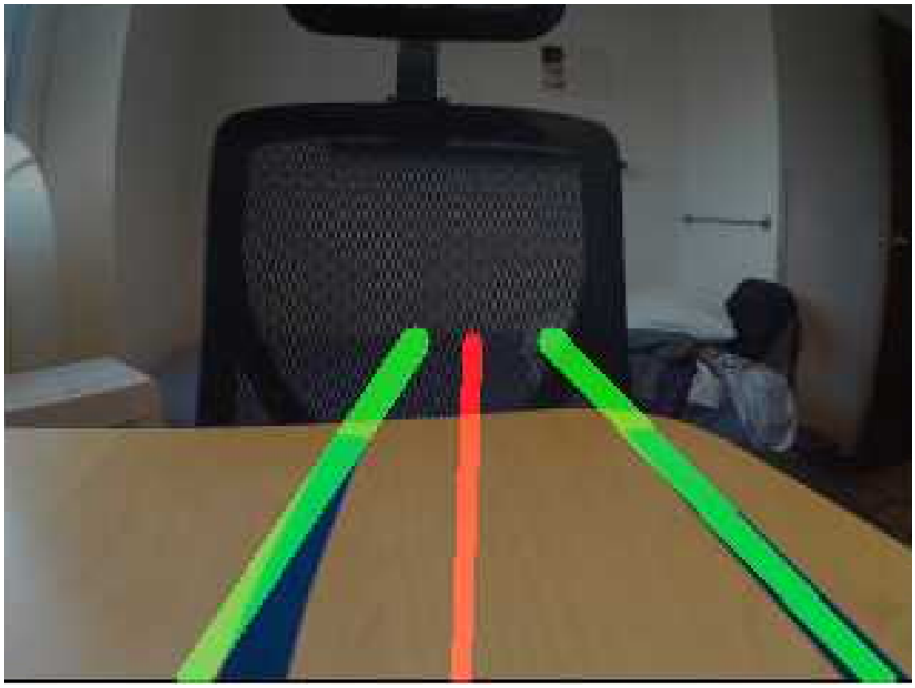


Figura 4.6: Linea de dirección

4.3.5. Estabilización

A veces el vehículo hace giros rápidos corrigiendo la dirección de forma brusca. Ese no es un comportamiento humano que queramos replicar.

Se ha visto que la Raspberry pi 3 procesa del entre 5 y 10 fotogramas por segundo y a veces la dirección cambiaba bruscamente.

Al cambiar a la Raspberry Pi 4, el procesamiento es 10 veces más rápido de media,con lo que esto ocurre en menor medida.

Finalmente tras muchas pruebas se ha visto que el comportamiento sin la función de estabilización con la Raspberry Pi 4 haga que el vehiculo pierda menos el carril. Por lo tanto se ha decidio no usarlo. El código asociado a esta estabilización es la función stabilize_steering_angle dentro del archivo hand_coded_lane_follower.py

4.4. Uniendo todo lo anterior

Toda este código se une en hand_coded_lane_follower.py **?**

Capítulo 5

Navegación. Conducción autónoma con aprendizaje profundo

5.1. Introducción

En este capítulo se va a entrenar el vehículo para que aprenda a mantenerse dentro del carril de forma autónoma. Para ello vamos a usar la lógica del capítulo anterior, usando los vídeos del circuito y el etiquetado de los ángulos de giro para da fotograma como entrada de datos. Se va a usar un modelo basado en el proyecto NVIDIA's DAVE-2, que usó NVIDIA para entrenar un vehículo autónomo de tamaño normal aprendiendo de horas de conducción manual. Este proyecto usa redes neuronales convolucionales para detectar los elementos de la carretera y girar acorde a ellos.

Esta aproximación es parecida a cómo los humanos aprendemos a conducir. Observamos a otros hacerlo y aprendemos cometiendo nuestros propios errores.

Hasta ahora, se han seguido los pasos necesarios para hacer este coche seguir un carril. Aislar el color, detección de bordes, detección de líneas, giros, estabilización de la suavidad de los giros... etc)

Además se tienen que configurar algunos parámetros de configuración manual, como los rangos máximos y mínimos del color azul de la cinta que marca los carriles, algunos parámetros para detectar segmentos con la transformada de Hough y el máximo giro de las ruedas directrices.

Llegar a una configuración que haga que el vehículo se mueva suave en los carriles, es una tarea larga y compleja.

Esto es un problema ya que cada vez que se tengan diferentes condiciones de circuito o circuitos diferentes tendremos que reconfigurar esos valores o incluso retocar los algoritmos.

Vamos a basarnos en un paper de nVidia que enseña a aprender un coche a escala real a conducir. Bojarski et al. (2016)

5.2. Redes neuronales convolucionales

Una red neuronal convolucional es un tipo de red neuronal que tiene la capacidad de descifrar patrones complejos en enormes conjuntos de datos de imágenes.

Con esta herramienta el vehículo puede percibir el mundo que le rodea.

Este tipo de redes ha sido diseñado para estudiar la estructura espacial de las imágenes.

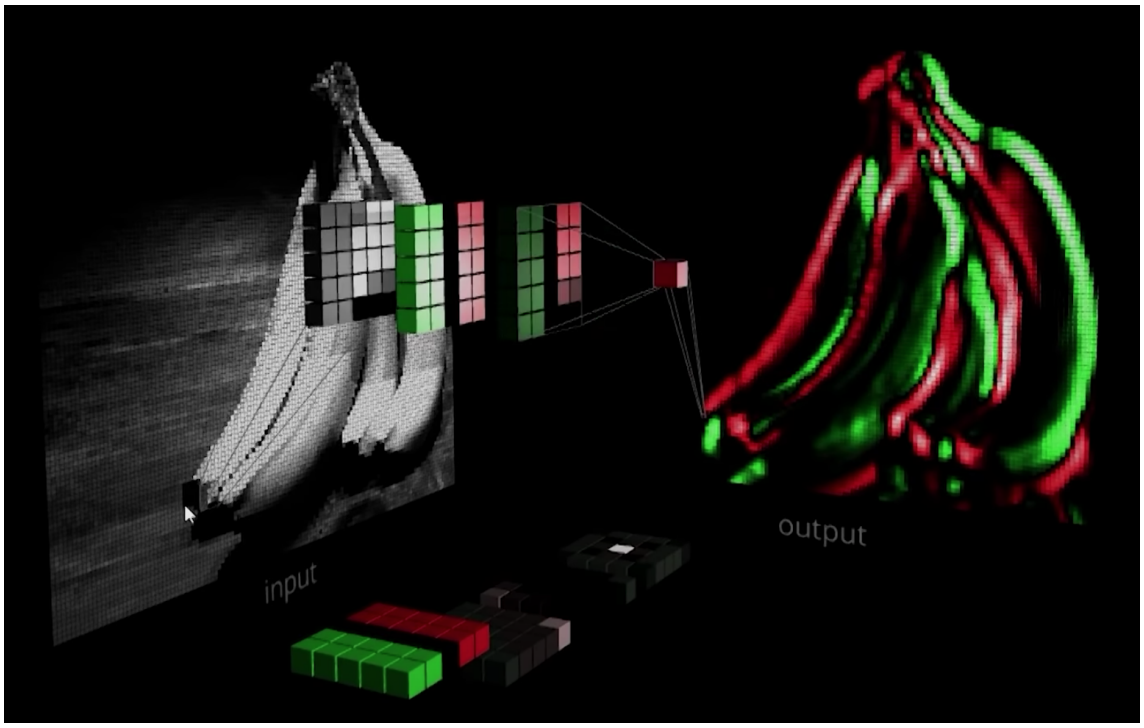


Figura 5.1: Red neuronal convolucional

Una red neuronal convolucional se caracteriza por aplicar un tipo de capa donde se realiza una operación matemática conocida como convolución

Este tipo de redes es el más usado en el reconocimiento de imágenes para modelos de aprendizaje profundo. Este tipo de redes son muy buenas extrayendo elementos visuales característicos de imágenes en varias capas llamadas filtros.

De esta manera el vehículo puede percibir el mundo que le rodea. Este tipo de redes ha sido diseñado para estudiar su estructura espacial.

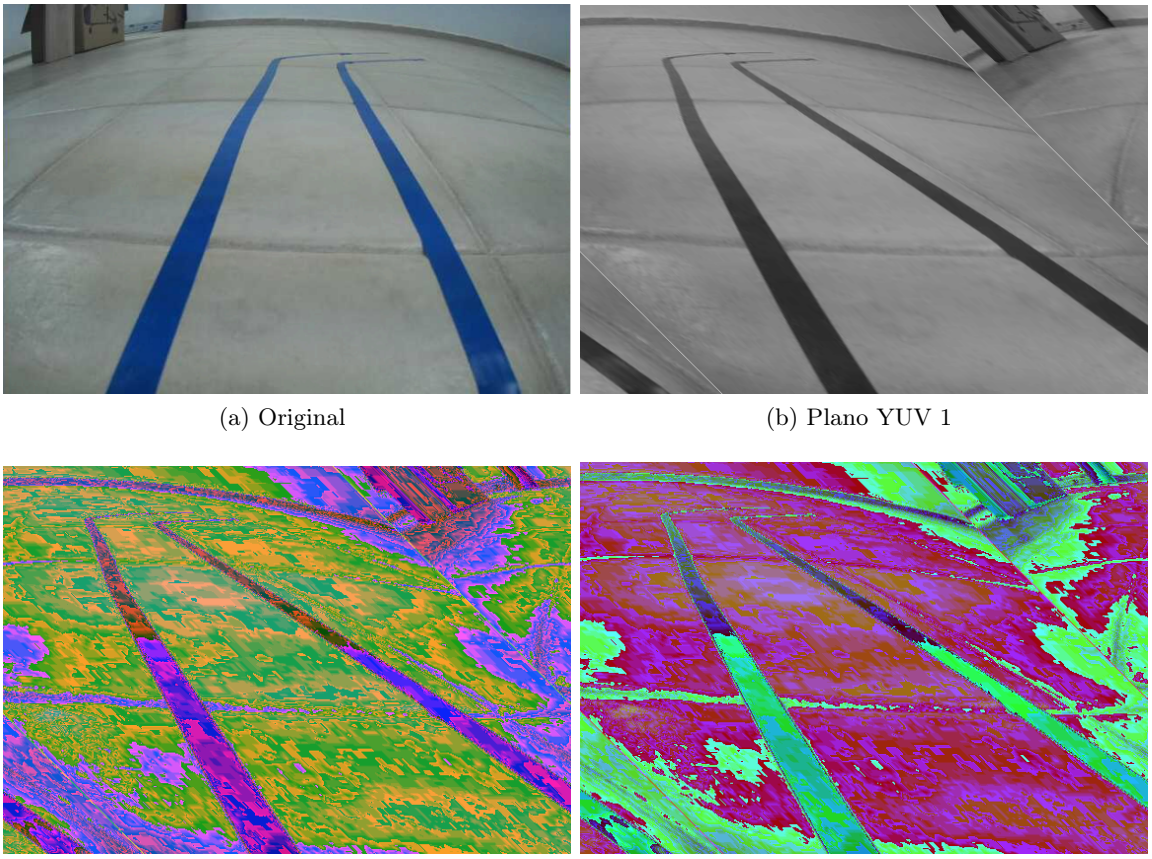
5.3. Modelo Nvidia

Esencialmente el modelo NVidia es una red neuronal convolucional cuya entrada al modelo son los fotogramas del video de la cámara del vehículo y la salida es el ángulo de giro de las ruedas directrices.

Como ya conocemos en aprendizaje supervisado, las imágenes de entrada se conocen como features y el ángulo de salida se conoce como Labels y se usan para el entrenamiento.

Como los valores del ángulo son numéricos, se trata de un problema de regresión.

Las capas de la red neuronal convolucional en el modelo NVidia extrae lineas y bordes



(a) Original (b) Plano YUV 1

(c) Plano YUV 2 (d) Plano YUV 3

Figura 5.2: Planos YUV de entrada a la red NVidia

así como capas y texturas en sus ultimas capas. Las capas totalmente conectadas a la salida de la red neuronal funcionan como controlador de giro.

El modelo NVidia tiene 30 capas en total. 5.3 La imagen de entrada, en la parte ingerior del diagrama, es de 66x220px. Primero se normaliza, después 5 grupos de capas convolucionales y luego 4 capas totalmente conectadas para salir a una capa única de salida que es el ángulo de giro de las ruedas directrices.

La imagen de entrada se divide en planos YUV antes de pasar por la red.

YuV es un espacio de color usado habitualmente como parte de un sistema de procesamiento de imagen en color. Esta forma de transmitir las imágenes hace que las imperecciones de compresion se oculten mas eficientemente que una representación RGB. Las imágenes serían algo parecido a lo que se puede ver en la 5.2

?

El ángulo predicho por el modelo se compara con el ángulo dado por el vídeo, y el error vuelve a alimentar la red para entrenar el proceso mediante retropropagación (back propagation).

El proceso es bastante típico, excepto por la salida que suele ser un tipo de clasificación y en este caso es un valor numérico.

En la red de NVidia se han entrenado los pesos para minimizar el error entre el ángulo

de salida de la red y el ángulo de salida registrado por el humano.

En la figura 5.3 se puede ver un esquema de las capas del modelo de red neuronal convolucional de nVidia.

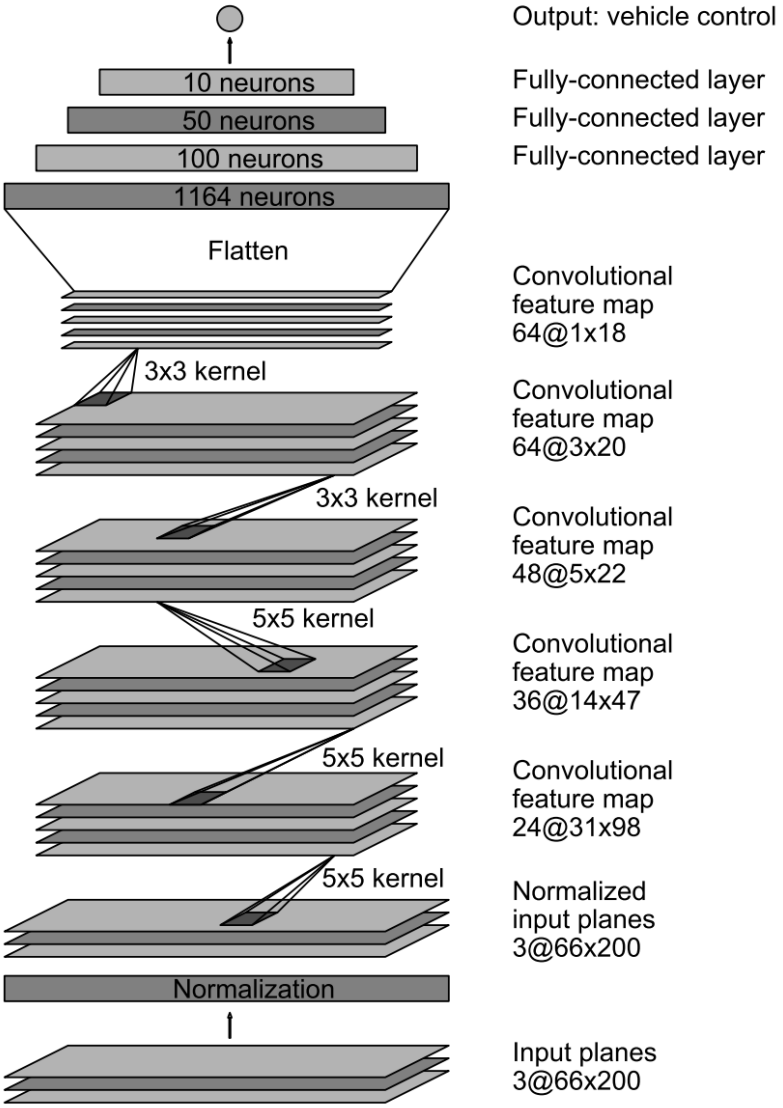


Figura 5.3: Modelo de red neuronal convolucional nVidia

5.4. Adaptando el modelo al proyecto

Además del tamaño, nuestro coche es bastante similar al de NVidia. NVidia obtuvo sus datos con 70 horas de conducción humana. Nuestro coche necesita obtener algunas imágenes de un vídeo y grabar el correcto ángulo de giro de las ruedas directrices.

5.4.1. Adquisición de datos

Hay muchas formas de adquirir esos datos.

Una de estas formas es escribir un programa de control remoto y que aprenda del comportamiento de conductores humanos. Esta es la mejor manera de simular un comportamiento humano. Se ha utilizado un programa de control remoto usado en este máster pero ha resultado ser lento para representar un comportamiento suficientemente fiable.

Otra es usar el código para mantener el vehículo dentro del carril usando OpenCV del capítulo anterior y así hacer que una máquina aprenda de otra. Solo tenemos que abrir nuestro çonductorz guardar el vídeo y los correspondientes ángulos de giro.

Se ejecuta el código asociado a drivers/code/save_image_and_steering_angle.py de un video y genera los fotogramas con su asociación al ángulo de giro. Se debe ejecutar el comando sin la extensión del video (.avi). El resultado es algo como lo se ve en la imagen 5.4

Listing 5.1: save_image_and_steering_angle.py

```
pi@raspberrypi:~/TFMIOT/DeepPiCar-master/driver/code $ python3
    save_training_data.py ../../models/lane_navigation/data/images/
    video01
```

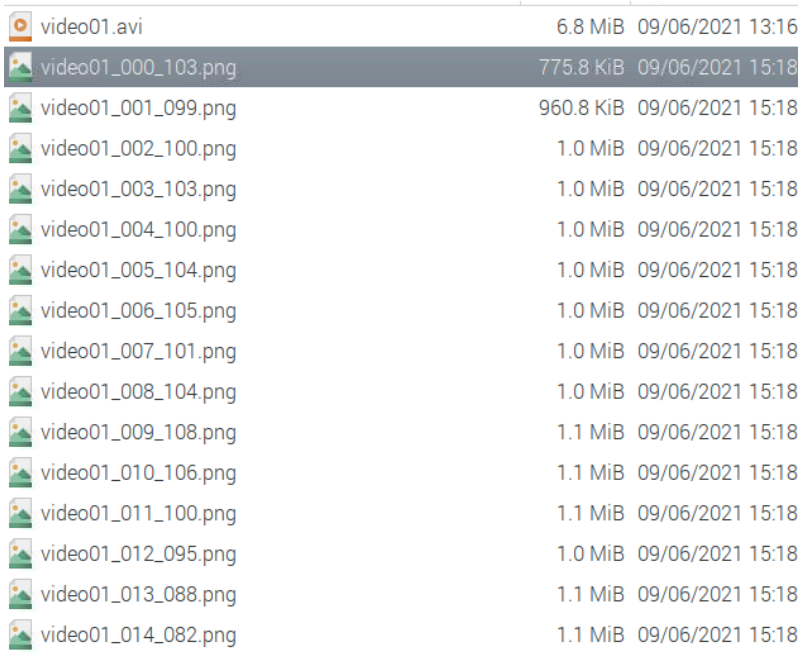


Figura 5.4: Fotogramas a partir del video con el angulo asociado

También "deep_pi_car.py"guardará automáticamente un vídeo cada vez que se ejecute.

5.4.2. Entrenamiento. Aprendizaje profundo

Ahora que se han obtenido las features, fotogramas del directo, y las labels, ángulos de giro, se puede aplicar aprendizaje profundo.

Hay que remarcar que a pesar de la moda del aprendizaje profundo, esta parte es solo una pequeña parte del proyecto.

La mayor parte del tiempo se ha ido en afinar y mejorar el hardware, aspectos de

software, limpieza de datos, retocar código etc...

Para el entrenamiento del modelo es muy lento usar la CPU de la raspberry pi, necesitaremos una GPU.

Por eso se va a usar Google Colab que permite usar GPU o TPU gratuitamente.

¿Que es Google Colab?

Google Colab es un proycto de Google que permite escribir y ejecutar codigo Python en el navegador. Está diseñado para tareas de aprendizaje automático y análisis de datos. Es un sistema de Jupyter Netbook integrado en web que permite usar los recursos de ejecución de Google, como GPU's o TPU's.

El código asociado, que está inspirado en el artículo Bojarski et al. (2016), se encuentra en end_to_end_lane_navigation.ipynb. Se va a proceder a comentar los pasos de este entrenamiento.

5.4.3. Cargar datos de entrenamiento

Se van a cargar los daros de entrenamiento desde la carpeta images. Esta carpeta se encuentar en el subida al repositorio de GitHub un data set de imágenes /models/lane_navigation/data/images/

El nombre de cada archivo es videoXX_FFF_SSSS.png, donde cada elemento es:

- videoXX = nombre del video
- FFF = numero del frame en el video
- SSS = angulo en grados

El código asociado a esta en end_to_end_load_data.py.

5.4.4. División conjuntos Entrenamiento y Prueba

Para la división de los conjuntos de entrenamiento se ha usado la misma que se ha usado en el proyecto original 80/20.

5.4.5. Aumento del conjunto de imágenes

Como el número de imágenes que hemos usado para alimentar el conjunto de datos no son suficientes para entrenar un modelo de aprendizaje profundo, vamos a aplicar algunas operaciones sobre las imágenes originales para aumentar este conjunto de datos. Las operaciones que hemos usado de pre procesamiento son Zoom y Flip.

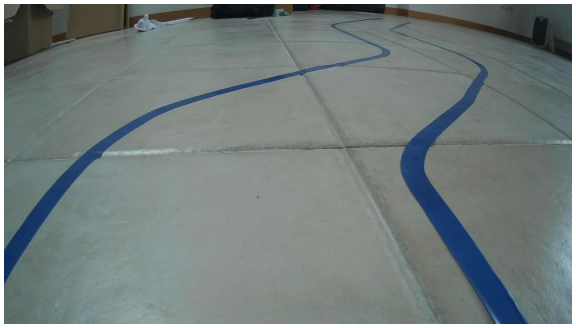
5.4.6. Zoom

A cada imagen se le ha aplicado un zoom aleatorio entre 100 % ->130 %. El código asociado se encuentra en la función zoom de end_to_end_zoom.py

(a) Imagen Entrenamiento 1

(b) Imagen Entrenamiento 2

(c) Imagen Entrenamiento 3



(d) Imagen Entrenamiento 4

Figura 5.5: Ejemplos de imágenes de entrenamiento

5.4.7. Flip

A cada imagen también se le ha aplicado un flip aleatorio. El código asociado se encuentra en la función zoom de end_to_end_flip.py

5.4.8. Procesado de imágenes

Una vez se tiene un conjunto de imágenes considerable, necesitan ser procesadas para que sean compatibles con la entrada de imágenes del modelo NVidia.

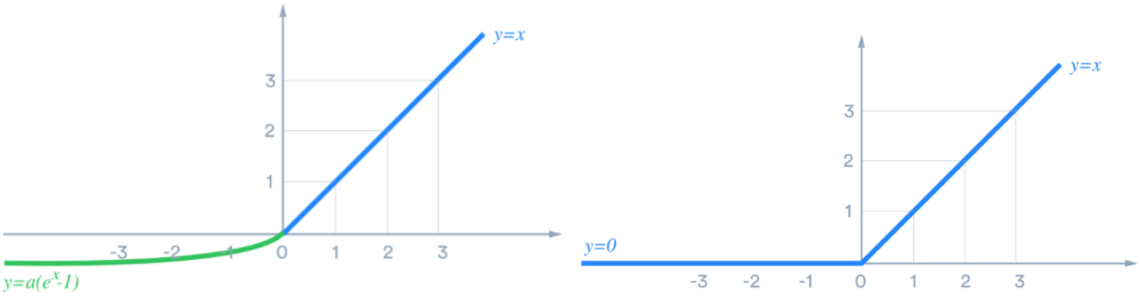
El modelo que usó NVidia, se puede consultar en el siguiente artículo Bojarski et al. (2016). Este artículo llamado End to End Learning for self-driving cars muestran como aprende un vehículo autónomo entrenando una red neuronal convolucional desde las imágenes de unas cámaras frontales.

El modelo NVidia tiene una resolución de entrada de 200x66 pixels, por lo que finalmente se hace un redimiensionado de cada fotograma para ese tamaño. El código asociado está en end_to_end_img_preprocess.py

5.4.9. Modelo NVidia comparado con el modelo de Tian

El código asociado se encuentra en end_to_end_nvidia.py En la figura 5.3 se pueden observar las capas del modelo NVidia.

En el modelo implementado se han eliminado las capas de normalización, ya que se hace fuera del modelo. Se ha añadido una capa Dropout para hacer el modelo mas robusto.



(a) ELU (b) ReLU

La función de pérdida es Error cuadrático médio (Mean Squared error MSE) porque se está haciendo un entrenamiento de regresión.

Además se ha usado ELU (Exponential Linear unit) como función de activación en vez de ReLU (Rectifies Linear Unit) por que ELU no tiene problemas cuando x es negativa.

Cuando creamos el modelo y pintamos sus parámetros, vemos que contiene sobre 250.000 parámetros

```
Model: "Nvidia_Model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 31, 98, 24)        1824
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 47, 36)        21636
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 5, 22, 48)         43248
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 20, 64)         27712
_________________________________________________________________
dropout (Dropout)            (None, 3, 20, 64)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 1, 18, 64)         36928
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1152)              0
_________________________________________________________________
dense (Dense)                (None, 100)               115300
_________________________________________________________________
dense_1 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_2 (Dense)              (None, 10)                510
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 11
=================================================================
Total params: 252,219 Trainable params: 252,219 Non-trainable
params: 0
```

Figura 5.6: Parámetros nVidia

5.4.10. Resultados del entrenamiento

Ahora que el modelo los datos están listos, se entrena el modelo. El código asociado está en end_to_end_train.py

Normalmete se usa en keras model.fit() pero esta vez se ha usado model.fit_generator() porque no se usa un conjunto de datos de entrenamiento estático, sino que se genera dinámicamente del conjunto de datos aleatoriamente.

Esto devuelve un conjunto nuevo en cada iteración.

5.4.11. Evaluación del modelo entrenado. Resultados en carretera

Tras entrenar 30 minutos, el modelo ha terminado 10 épocas. Primero va a pintar la función de pérdida de los conjuntos de entrenamiento y validación.

Vemos que ambos descienden rápido y se mantienen abajo.

El código asociado está en end_to_end_loss.py

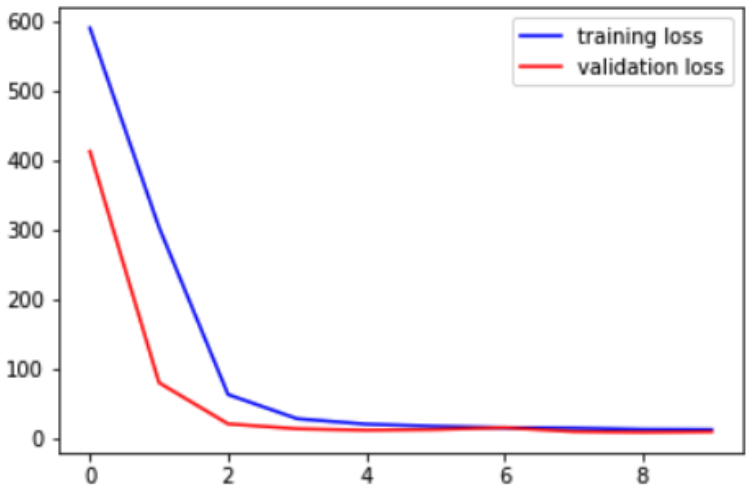


Figura 5.7: Funciones de pérdida y validación

Otra métrica que nos indica que el modelo ha ido bien es

$$R^2 \tag{5.1}$$

. Cuanto más cercana esté al 100 % mejor predicción, y se encuentra sobre un 93 %.

Capítulo 6

Reconocimiento de Objetos. Percepción.

6.1. Introducción

Actualmente casi todos los vehículos del mercado permiten adquirir un paquete que reconozca señales de tráfico o frenada en caso de atropello.

La idea de este capítulo es conseguir que el vehículo pueda percibir el mundo que le rodea. En concreto los elementos que le rodean que puedan hacer cambiar su comportamiento durante la circulación. Como por ejemplo señales de tráfico o peatones que pueda hacer detener el vehículo si se requiere o hacerle modificar su trayectoria.

Figura 6.1: Detección de señales

Para esto se va a hacer inferencia para usar el modelo de aprendizaje profundo MobileNet SSD en la TPU Edge Coral. Este capítulo está basado en el trabajo que Tian hizo y la idea es integrarlo en el vehículo.

La inferencia consiste en ejecutar un modelo ya entrenado con todo lo que necesita para resolver o clasificar el problema. Esta estructura permite que los procesadores de

inteligencia artificial aprendan estructuras complejas sin requerir grandes cantidades de datos.

6.2. Percepción

La detección de objetos es un problema típico para resolver con aprendizaje profundo. Hay dos componentes en un modelo de detección de objetos: la red neural base y la red neural de detección.

La red neural base son redes neuronales convolucionales que extraen características a bajo nivel, como lineas, bordes, o círculos o a alto nivel, caras, personas, señales de trafico.. etc.

Algunas redes neuronales convolucionales populares son LeNet, InceptionNet(GoogleNet), ResNet, ..etc

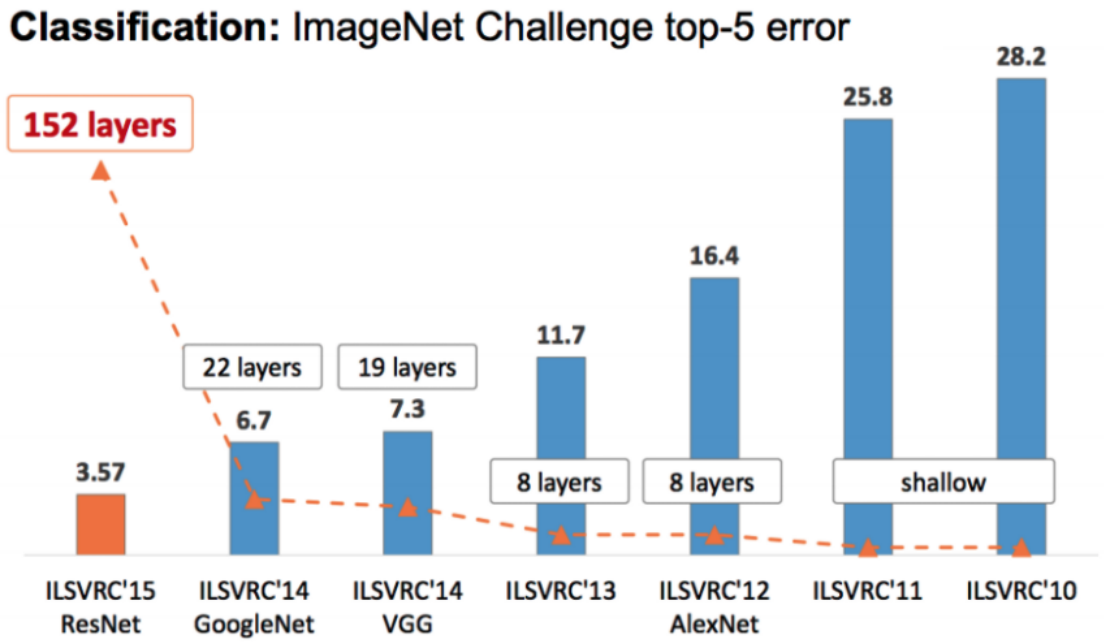


Figura 6.2: Numero de elementos detectados por Redes neuronales base

En este artículo se discuten las diferencias entre estas redes neuronales base Krizhevsky et al. (2012) y se hace una comparativa de rendimientos.

Las redes neuronales de detección se colocan al final de las redes neuronales base y se usan para identificar múltiples objetos de una imagen con la ayuda de las características de salida.

Algunas populares son SSD, R-CNN, Daster R-CNN, YOLO... etc

En este articulo se discuten las diferencias entre ellas Shanahan y Dai (2019) y en el siguiente artículo Liu et al. (2016) se presenta un método para detectar objeto a partir de imágenes usando redes neuronales.

El nombre de un modelo de detección de objetos normalmente viene dado por una combinación de su red base y su red de detección. Por ejemplo: "MobileNet SSD.º Ïncepcion SSD.º ResNet Faster R-CNN".

Por otro lado para modelos de detección pre-entrenados, el nombre del modelo también incluye el conjunto de datos de imágenes con el que fue entrenado.

Algunos conjuntos de datos famosos son son COCO dataset(100 objetos comunes), Open Images dataset(200000 tupos de animales y planteas) y iNaturalist dataset(200000 tipos de animales y plantas)

Por lo tanto el nomre ssd_mobilenet_v2_coco usa la segunda versión de MobileNet para la extracción de características, SSD para la detección de objetos y COCO como conjunto de datos pre-entrenado.

En este enlace hay una lista de modelos pre-entrenados con TensorFlow llamado Model Zoo.Yu et al. (2020)

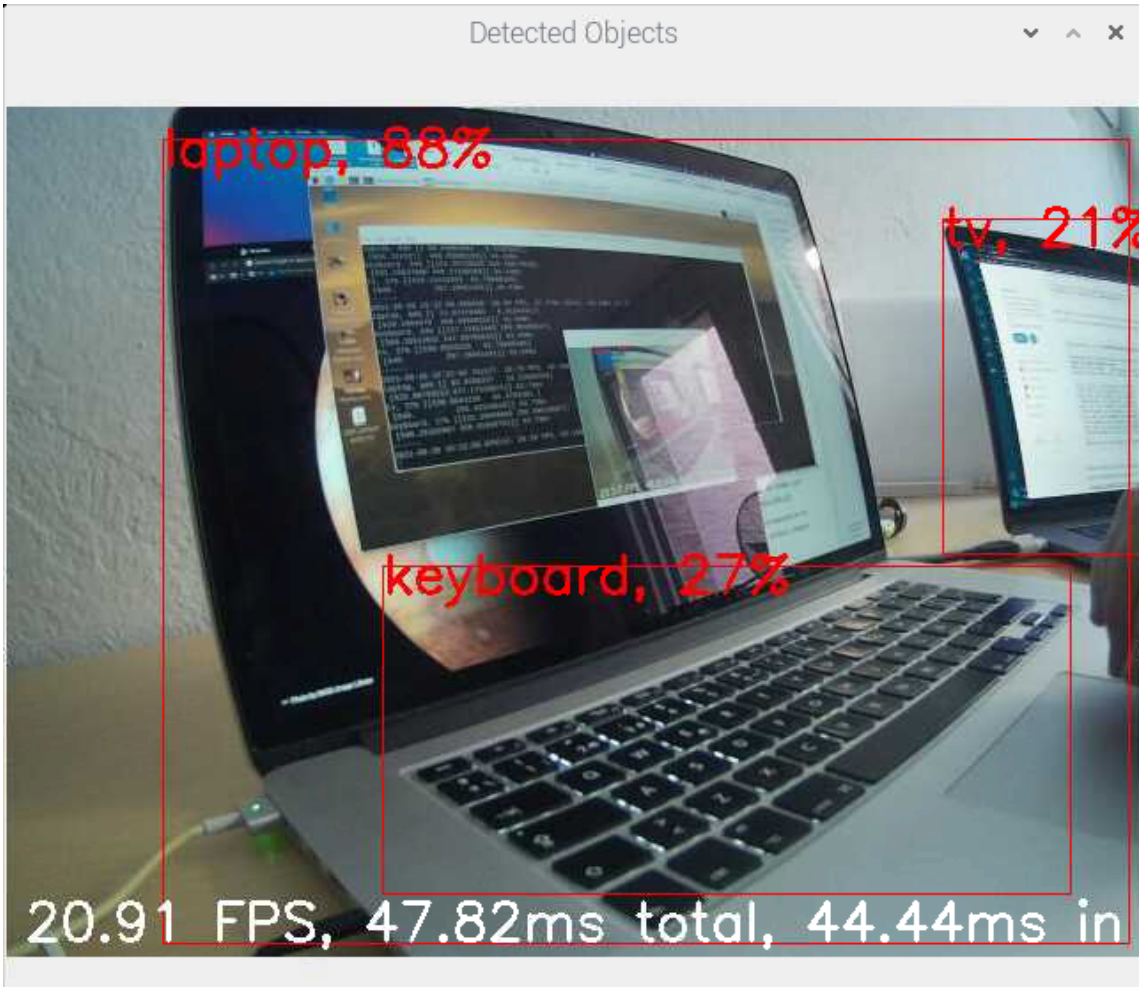


Figura 6.3: Elementos detectados COCO

6.2.1. Transfer Learning

En realidad lo que se quiere es detectar unas señales de trafico de juguete y algún playmovil a modo de peatón, no nuestros libros, ordenadores, móviles... etc.

No queremos etiquetar cientos de miles de imágenes y perder semanas o meses entrenando un modelo de detección profunda de cero. Lo que podemos hacer es aprovechar Transfer

Learning, que empieza con los parámetros de un modelo pre-entrenado, con 50 o 100 de nuestras propias imágenes y etiquetas y solo dedicarle unas pocas horas al entrenamiento.

REVISAR ESTA PARTE

REVISAR ESTA PARTE

REVISAR ESTA PARTE

6.2.2. Model Training

Los pasos a seguir para el entrenamiento del modelo son:

1. Obtención de imágenes y etiquetado
2. Selección del modelo
3. Transfer Learning / Model trainig
4. Guardar la salida del modelo en el formato valido para la Edge TPU
5. Inferir el modelo en la Raspberry Pi.

6.2.3. Selección del modelo

Una Raspberry Pi tiene poca capacidad de computación por eso se debe usar un modelo relativamente rápido y preciso. El modelo MobileNet v2 SSD COCO como el mas equilibrado para la potencia de la Raspberry Pi.

Para el modelo de Edge TPU se debe elegir el modelo MobileNet v2 SSD COCO Quiantized.

La cuantificación, que se puede ver explicada en el siguiente enlace: Krishnamoorthi (2018), es una forma de hacer inferencia de modelos mas rápido. Para ello se guardan los parámetros del modelo con valores discretos, en vez de con valores discretos, con muy poca degradación en la precisión de la predicción.

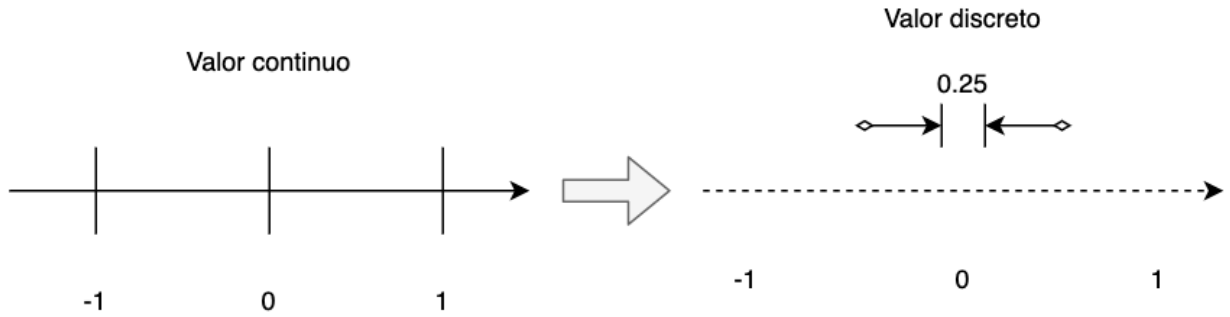


Figura 6.4: Cuantificación

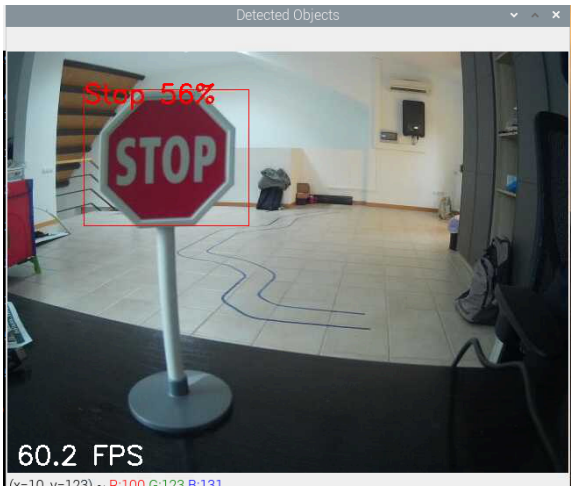
La Edge TPU está optimizada y solo puede rodar modelos cuantificados.

Kist (2021) En este articulo se profundiza mas en el rendimiento del Edge TPU.

Para esto usaremos Google Colab de nuevo. Este trabajo se va a basar en el trabajo Chengwei. En la siguiente url se puede ver tutorial: Zhang

El código asociado está en tensorflow_traffic_sign_detection.ipynb

(a) Peatón



(b) Stop

6.2.3.1. Preparación del entorno.

Elige el modelo MobileNet v2 SSD COCO Quantized y descarga el modelo entrenado del repositorio GitHub de TensorFlow.

6.2.3.2. Preparando los datos de entrenamiento

Este código convierte las etiquetas xml en ficheros generados por la aplicación LabelImg a fomato binario (.record), asi TensorFlow puede procesarlo mas rápidamente.

6.2.3.3. Descarga del modelo pre-entrenado.

pretrained_model/model.ckpt

El código de arriba descarga los archivos del modelo pre-entrenado ssd_mobilenet_v2_quantized_300x30 y solo usara el archivo "model.ckpt"desde el que aplicará el "transfer learning"

6.2.3.4. Entrenar el modelo

Este paso es el que más tiempo consume, dependiendo del número de epocas puedes tomar varias horas.

Cuando el entrenamiento esté terminado veremos muchos archivos en el directorio model_dir. Buscaremos el último model.ckpt-xxxx.meta.

Durante el entrenamiento, podemos monitorizar el progreso de la función de pérdida y la precisión a traves de TensorBoard.

6.2.4. Probar el modelo entrenado

Tras el entrenamiento probamos con diferentes imágenes del conjunto de datos.

Como esperábamos se detectan suficientemente bien como para realizar las pruebas de conducción.

6.2.5. Guardar el modelo en la Edge TPU

Una vez entrenado exportamos el modelo meta al grafico de inferencia en formato Google ProtoBuf, después al formato de la Edge TPU.

Usamos el proceso seguido por David Tian en su proyecto.

Finalmente obtenemos road_signs_quantized.tflite que es compatible con dispositivos móviles y Raspberry PI pero aun no compatible con la Edge TPU.

Para ello tenemos que ejecutar el archivo con el compilador web de Coral. `https://coral.ai/docs/edgetpu/compiler/`

Tras subirlo, puedes descargar otro archivo con la misma extensión road_signs_quantized_edgetpu.tflite.

6.3. Gestión de objetos detectados

Ahora que el coche ya se identifican objetos hay que decirle que queremos que haga cuando detecta cada uno de ellos.

REVISAR ESTA PARTE
REVISAR ESTA PARTE
REVISAR ESTA PARTE

Hay dos aproximaciones:

1. Basado en reglas. Una es decirle al coche exactamente lo queremos que haga cuando detecta algo. Esto es parecido a lo que hicimos en el reconocimiento de objetos con OpenCV.

2. Basado en punto a punto. Alimentar al coche con muchas horas de vídeo hasta que el coche aprenda que debe parar en un Stop o cuando vea un peatón. Parecido en lo que hicimos en el apartado anterior.

Se va a proceder con la primera ya que es como humanos aprendemos primero con las reglas de circulación, no a base de prueba y error. Y segundo porque es mas sencillo de implementar.

El código asociado a estas reglas se encuentra en: traffic_objects.py y object_on_road_processor.py

6.4. Rodando todo junto

Como se puede ver en la imagen 6.5 el vehículo es capaz de procesar el reconocimiento de los límites del carril y la detección de los elementos por encima de los 40 fotogramas por segundo.

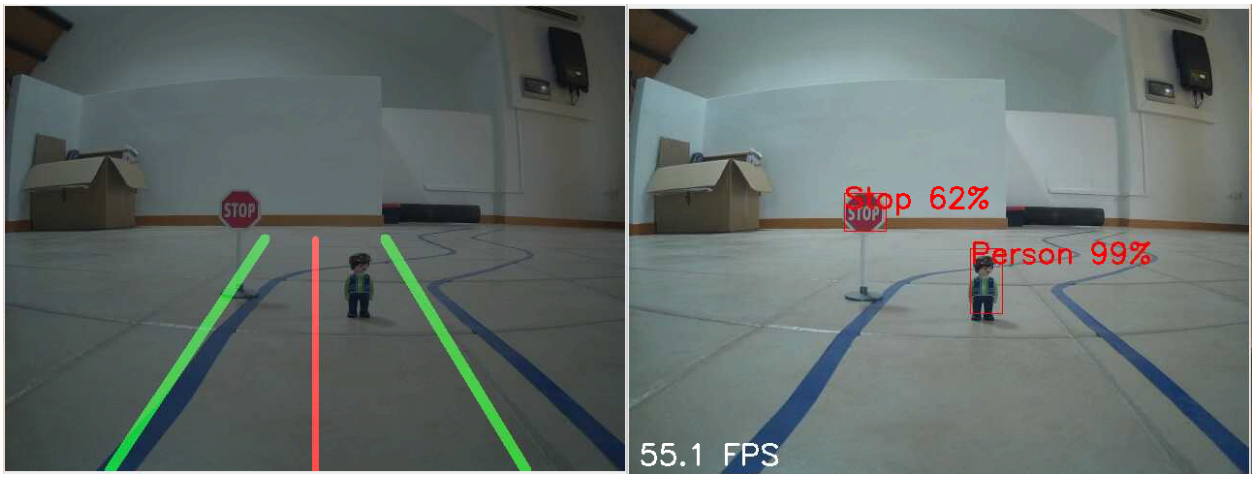


Figura 6.5: Rodando todo junto

Capítulo 7

Resultados

Es difícil medir los resultados cuando el proyecto ha pasado por tantas partes. Cada capítulo ha tenido su dificultad y pequeños detalles que han consumido mucho tiempo.

Los objetivos que se pusieron al inicio del proyecto fueron:

- Implementar un vehículo que sea capaz de circular por un carril de forma autónoma
- Que detecte y reaccione a elementos de su entorno, como peatones y señales de tráfico
- Capaz de aprender nuevos elementos a los que reaccionar

Al final el proyecto se ha conseguido tener un vehículo capaz de circular por un carril de forma autónoma. Reacciona a los elementos del entorno y es capaz de aprender nuevos elementos siempre y cuando se le entrene para ello.

Funciona como una una base libre y económica para futuros desarrollos donde poder hacer pruebas y desarrollar nuevos sistemas de conducción autónoma. De hecho queda en trabajo futuro muchas ideas que han ido surgiendo durante la ejecución de este proyecto.

Dista mucho de un nivel cuatro de autonomía como se pretendía en un principio, pero funciona bien como entorno de pruebas.

Otro objetivo era utilizar la inteligencia artificial y otros elementos aprendidos en el máster para llevar a cabo estas tareas. Finalmente se ha llevado a cabo. Se ha echado en falta más horas de entrenamiento.

La mayor parte de las horas se han dedicado a problemas secundarios que no han aportado valor al proyecto en si. Los problemas de las actualizaciones de la versión de Python y alimentación del nuevo hardware han dejado mucho esfuerzo que no se verá reflejado.

Pero el Resultado de ver mejorado el procesamiento de imágenes de 5 fotogramas por segundo a una media de 40, ha sido un gran logro.

Por otro lado no se ha conseguido mejorar significativamente el comportamiento que se tenia inicialmente en el entrenamiento que hizo David Tian de su percepción de señales de tráfico.

Seria usar más de un sistema de entrenamiento para el reconocimiento de los limites del carril, como por ejemplo alguna aplicación móvil manual por Bluetooth. Ya que los errores de comportamiento del software de conducción autónoma se han llevado al modelo.

El resultado final es mejorable en cada capítulo pero en conjunto hace un entorno de desarrollo y un vehículo autónomo como se pensó al inicio del proyecto.

Capítulo 8

Conclusiones y Trabajo Futuro

Como trabajo futuro se han abierto muchas opciones. Como el proyecto abarca casi todos los aspectos desde el diseño, el hardware, hasta el reconocimiento de objectos, en cada capítulo se han observado situaciones que pueden quedar para un apartado de trabajo futuro.

Primero se tendría que trabajar en la batería. La Raspberry Pi 4 y la Unidad central de tensores suponen un consumo extra de alimentación que es difícil de suplir. Se ha hecho con dos baterías. Este es un gran punto de mejora. Una batería con menor capacidad, pero con mayor potencia puntual.

Además una de las baterías entra en modo ahorro de energía y se para cuando considera que no se está usando. Y si el vehículo se detiene un tiempo considerable frente a un peatón por ejemplo, esta se apagará, teniendo que activarla con un botón manualmente.

Otro aspecto que queda para un trabajo futuro es terminar una aplicación de control remoto. Durante la realización del máster se hizo una aplicación que permitía controlar el vehículo via Bluetooth y MQTT. Pero no se ha conseguido hacer ejecutar al mismo tiempo que el software de conducción autónoma y detección de objetos.

Por ultimo el entrenamiento que se ha hecho puede ser mejorado con mejores vídeos que sean usando otro software de etiquetado que no sea el visto en el capítulo 4. Esto puede dar mayor variedad en el aprendizaje y hacer que el vehículo se comporte mejor.

El aspecto más crítico a mejorar puede ser el de la temperatura. A pesar de haber incluido un disipador y un ventilador en el vehículo, este se calienta y a veces hace que el procesamiento de los fotogramas baje temporalmente. Este aspecto queda como trabajo futuro también.

En conclusión, todos los capítulos han tenido sus dificultades y visto en conjunto mejorarlas puede aportar mayor calidad al conjunto.

Chapter 9

Introduction

Currently all car manufacturers are in a battle for safety and make their vehicles more autonous. Each one allows you to purchase different autonomous driving packages. Depending on the degree of autonomy, each package allows different functions.

What is autonomous driving? According to the SAE Society of Automotive Engenieers there are different types of levels when classifying an autonomous vehicle as you can see in the following figure 9.1 as we can see in its website SAE

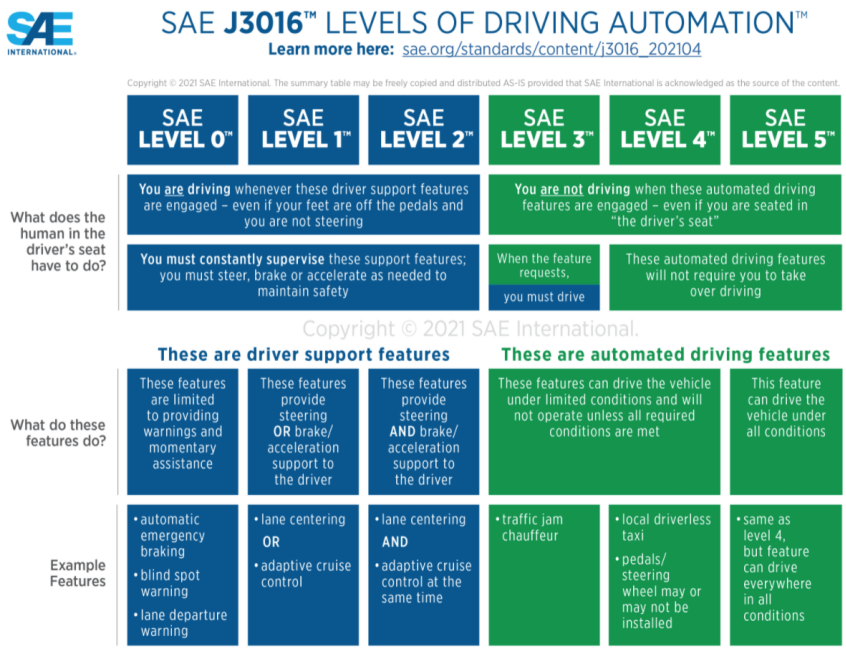


Figure 9.1: SAE autonous levels

This scale classifies the autonomous capabilities of a vehicle into six different levels. Level zero would be a car that does not have any assisted driving functionality. Levels one and two describe a type of driving in which certain critical tasks such as accelerating, braking or steering are partially controlled by the system, with human supervision. At level three in certain cases it would not require continuous monitoring. At level four in

certain limited scenarios, human supervision would be dispensable, for example main roads with favorable weather. If what was achieved by level four were possible in any scenario, we would be talking about level five. Level five is a level of full autonomy.

Currently the market of vehicle manufacturers is marked by the development of technology to improve consumption efficiency or aid in driving. Such is the point where there is a shortage in the world of semiconductors that can directly affect this sector.

In the project that you want to carry out, it would fall within level four, since in a certain circuit and with certain elements it intends to be completely autonomous.

Currently on the market you can buy different autonomous driving packages. Each company has its names. The Tesla website has been taken as a reference.

Tesla

On this page you can see the different types of driving assistance that your vehicles include.

- Recognition of the speed of the vehicle in front
- Helps to follow the lane by turning the steering wheel
- They have Autopilot navigation in beta, which, giving you a route, takes you on the highway from the entrance to the highway exit.
- Automatic lane change
- Parking assistance
- Allows you to move the car with the Tesla application
- Recognition of traffic lights and signals
- Automatic emergency brake
- Front and side collision alarm
- Adaptive Cruise Control (ACC)
- Lane Keep Assist System (LKAS)

These technologies that assist driving aim to detect and recognize the environment around a vehicle.

This recognition can be done with multiple sensors such as cameras, radars, lasers or ultrasounds, among others.

These sensors allow to form more complex systems that can be mounted in a modular way in any vehicle. This involves many technical legal and moral aspects. Artificial intelligence has been the key to all these features. So much so that all manufacturers allow, to a greater or lesser extent, the integration of systems that help driving intelligently. It is even probable that soon we will see laws associated with the driving of vehicles by artificial intelligences.

9.1. Art studio

This project is a study on autonomous vehicles. For this, it has been decided to make a low-cost vehicle that serves as a development environment. In this environment, all the projects and sensors are going to be integrated in order to achieve the widest possible autonomous driving. It is also intended to leave an open environment for the future integration of new technologies and compatible sensors.

In this vehicle it is intended to use artificial intelligence. For this, a camera has been integrated with which it is intended that the vehicle recognizes its surroundings and can make decisions regarding them.

They also want to use all the available tools that can speed up the integration process.

Perhaps this project can serve as a preliminary environment to train the technologies of the autonomous vehicles of the future.

We have found numerous autonomous vehicle projects on which we could base our project. Getting started requires finding a base vehicle that provides basic movements and vehicle control capable of at least accelerating, braking, and turning.

Among all the vehicles that have been studied, the simplest and most economical that has been found is the Sunfounder PiCar-S model. In addition, the processing base is a Raspberry Pi that provides us with a lot of configuration potential and upgrade possibilities.

Studying the projects that have worked on autonomous vehicles, some have been found with the same purposes as cite deeppicar2018. What does a low-cost autonomous vehicle based on deep learning do?

But there is one in particular that has used a similar base vehicle. It also shares a base vehicle control API.

It is the David Tian DeepPiCar project. This project uses a similar model, the PiSar-V. This model is equipped with a mechanized camera on the front capable of moving to follow the direction of the lane.

Both vehicles share a core in the library that allows turning and setting the vehicle speed.

The project we want to do is integrate a large part of the DeepPicar project by adapting it to the PiCar-S vehicle.

In addition, they want to update the Raspbery Pi for a model with greater processing capacity.

The DeepPiCar project is programmed with version 2 of Python, a version that was withdrawn on January 1, 2020, so the project has been made compatible with version 3.

9.2. Motivation

The purpose of this project is to create a low-cost autonomous vehicle using open source tools to learn the specific operation of any of these systems listed in the previous section.

Given the complexity of the project, all the means that we have available and open source will be used and it will be based on the approaches that other people have previously made.

The idea is that the result of this project can be mounted independently in any vehicle and can be used as a development environment to test future sensors or technologies that can help autonomous driving without making a large financial outlay.

9.3. Objectives

The objective of this project is to make a vehicle capable of learning autonomous driving.

It is also intended to be a free and inexpensive basis for future developments where new autonomous driving systems can be tested and developed.

This project falls within level four of the levels proposed by the SAE, since with a specific type of circuit and with pre-established traffic events it is intended that the vehicle is completely autonomous.

The idea is that the vehicle applies one of these autonomous driving systems listed above.

Another objective is to use artificial intelligence and other elements learned in the master to carry out these tasks.

- Implement a vehicle that is capable of driving in a lane autonomously
- That detects and reacts to elements of its environment, such as pedestrians and traffic signals
- Be able to learn new items to react to

9.4. Way of working

The work plan to follow to achieve the objectives described in the previous section is described below.

- Find a vehicle base to host the project
- Make an art study
- Find and study similar projects
- Assemble and implement
- Study the results

9.5. Memory organization

This section lists and makes a short summary of the chapters that will be discussed throughout the project.

9.5.1. Hardware

All physical elements are covered in this section.

The challenge of assembling a low-cost miniature vehicle that allows one of the systems seen above to be assembled.

Improvements and optimizations as a result of learning from carrying out this project.

We will list and describe all the hardware elements that we will need for the execution of the project.

9.5.2. Software

This section covers the pieces of software that we need.

Open source projects will be sought that allow us to accelerate the work and adapt it to the needs of the project.

We will improve and integrate any solution we find that allows us the purpose of the project.

A tour will be made to install the project from scratch on the Hardware described in the previous section.

9.5.3. DeepPiCar project

A similar project that can be adapted to the requirements of this project will be analyzed.

The project will be adapted to this vehicle.

9.5.4. Autonomous driving

In this section, software will be adapted to drive within the lane limits.

This driving will use the camera only.

This section does not use artificial intelligence.

9.5.5. Training and results

Training and results The software from the previous section will be used to train an artificial intelligence to learn to drive within the limits of a lane.

9.5.6. Object recognition

Other artificial intelligence will be used to recognize objects that can modify the behavior of the vehicle while driving.

Chapter 10

Conclusions and Future Work

As future work, many options have been opened. As the project covers almost all aspects from the design, the hardware, to the recognition of objects, in each chapter situations have been observed that may remain for a section of future work.

First you would have to work on the drums. The Raspberry Pi 4 and the Central Tensioner Unit represent extra power consumption that is difficult to supply. It has been done with two batteries. This is a great point of improvement. A battery with a lower capacity, but with higher point power.

In addition, one of the batteries enters energy saving mode and stops when it considers that it is not being used. And if the vehicle stops for a considerable time in front of a pedestrian for example, it will turn off, having to activate it with a button manually.

Another aspect left for future work is finishing a remote control application. During the completion of the master, an application was made that allowed the vehicle to be controlled via Bluetooth and MQTT. But it has not been able to run at the same time as the autonomous driving and object detection software.

Finally, the training that has been done can be improved with better videos that are using other labeling software than the one seen in chapter 4. This can give more variety in the learning and make the vehicle behave better.

The most critical aspect to improve may be that of temperature. Despite having included a heatsink and fan in the vehicle, the vehicle heats up and sometimes causes frame processing to temporarily slow down. This aspect remains as future work as well.

In conclusion, all the chapters have had their difficulties and, seen as a whole, improving them can bring greater quality to the whole.

Bibliografía

Bechtel, M., Mcellhiney, E., Kim, M. y Yun, H. Deeppicar: A low-cost deep neural network-based autonomous car. páginas 11–21. 2018.

Bojarski, M., del Testa, D. W., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J. y Zieba, K. End to end learning for self-driving cars. *ArXiv*, vol. abs/1604.07316, 2016.

Duda, R. y Hart, P. Use of the hough transform to detect lines and curves in pictures. *Communications of The ACM - CACM*, vol. 15, 1975.

Intel. *Open Source Computer Vision Library*. 2013.

Kist, A. Deep learning on edge tpus. 2021.

Krishnamoorthi, R. Quantizing deep convolutional networks for efficient inference: A whitepaper. *ArXiv*, vol. abs/1806.08342, 2018.

Krizhevsky, A., Sutskever, I. y Hinton, G. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, vol. 25, 2012.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C.-Y. y Berg, A. C. Ssd: Single shot multibox detector. En *ECCV*. 2016.

Podpora, M., Korba´s, G. y Kawala-Janik, A. Yuv vs rgb – choosing a color space for human-machine interaction. *Annals of Computer Science and Information Systems*, vol. Vol. 3, 2014.

SAE. Sae levels of driving automation™ refined for clarity and international audience. Disponible en `https://www.sae.org/blog/sae-j3016-update`.

Shanahan, J. y Dai, L. Realtime object detection via deep learning-based pipelines. páginas 2977–2978. 2019. ISBN 978-1-4503-6976-3.

Tesla. Tesla support. Disponible en `https://www.tesla.com/support/autopilot`.

Tian, D. Deeppicar — part 1: How to build a deep learning, self driving robotic car on a shoestring budget. Disponible en `https://towardsdatascience.com/deeppicar-part-1-102e03c83f2c`.

Yu, H., Chen, C., Du, X., Li, Y., Rashwan, A., Hou, L., Jin, P., Yang, F., Liu, F., Kim, J. y Li, J. TensorFlow Model Garden. `https://github.com/tensorflow/models`, 2020.

Zhang, C. Tutorial: How to train a custom object detection model easily with tensorflow object detection api and google colab's free gpu. Disponible en https://www.dlology.com/blog/how-to-train-an-object-detection-model-easy-for-free/.