Estudio y simulación de un vehículo autopilotado en Unity 5 haciendo uso de algoritmos de aprendizaje automático

Trabajo de fin de grado

Javier Antón Alonso
Xuebo Zhu Chen

Departamento de Arquitectura de Computadores y Automática

Facultad de Informática
Universidad Complutense Madrid

Curso académico 2017/2018

AUTORIZACIÓN PARA LA DIFUSIÓN DEL TRABAJO FIN DE GRADO Y SU DEPÓSITO EN EL REPOSITORIO INSTITUCIONAL E-PRINTS COMPLUTENSE

Los abajo firmantes, alumno/s y tutor/es del Trabajo Fin de Grado (TFG) en el Grado en
……de
la Facultad de ……..……, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el Trabajo Fin de Grado (TF) cuyos datos se detallan a continuación. Así mismo autorizan a la Universidad Complutense de Madrid a que sea depositado en acceso abierto en el repositorio institucional con el objeto de incrementar la difusión, uso e impacto del TFG en Internet y garantizar su preservación y acceso a largo plazo.

Periodo de embargo (opcional):
o 6 meses
o 12meses

TÍTULO del TFG: .....
…….
Curso académico: 20….. / 20…..

Nombre del Alumno/s:
……
……

Tutor/es del TFG y departamento al que pertenece:
……
……
……

Firma del alumno/s Firma del tutor/es

Memoria presentada para el trabajo de fin de grado de
Ingeniería Informática

Dirigida por
Sara Román Navarro

Departamento de Arquitectura de Computadores y Automática

Facultad de Informática
Universidad Complutense Madrid

Curso académico 2017/2018

Agradecimientos

Queremos agradecer los ánimos y la ayuda recibida a una gran cantidad de personas, gracias a las cuales este trabajo se ha podido llevar a cabo.

Primeramente , queremos agradecer a nuestra tutora Sara Román por darnos la oportunidad de realizar este TFG, por su buena fe y su disposición a los cambios realizados en el mismo .

Además queremos dar las gracias especialmente a nuestros familiares y amigos , que nos han dado fuerzas y prestado su ayuda siempre que nos hizo falta.

Agradecer también a todos los profesores que hemos tenido a lo largo de la carrera , que con su paciencia y su ayuda , nos han hecho adquirir los conocimientos necesarios en nuestra formación.

Resumen

En la actualidad, los vehículos autónomos han dejado de ser algo del futuro. Poco a poco se han logrado avances en este campo , hasta el punto de llegar a tener en funcionamiento taxis sin conductor .

En este proyecto hemos optado por implementar un sistema con el que experimentar diversas técnicas de aprendizaje automático en una simulación de un vehículo virtual autopilotado en Unity, basándonos en el modelo de NVIDIA "End to End".

Esto nos permite aumentar y poner en práctica los conocimientos obtenidos en asignaturas como aprendizaje automático .

Para llevar a cabo esto, el proyecto se divide en varias partes diferenciadas:

- Investigación y estudio (Nvidia model)
- Implementación del simulador (Unity 3D)
- Implementación scripts (Red Neuronal)
- Pruebas y conclusiones (Training & Testing)

Abstract

Currently, autonomous vehicles have ceased to be something of the future. Little by little, progress was made in this field, to the point of having taxis without driver in operation.

In this project we have chosen to implement a system with which to experiment with various machine learning techniques in a simulation of a virtual vehicle autopiloted in Unity, based on the NVIDIA "End to End" model.

This allows us to increase and put into practice the knowledge obtained in subjects such as machine learning.

To carry out this, the project is divided into several differentiated parts:

- Research and study (Nvidia model)
- Simulator implementation (Unity 3D)
- Scripts implementation (Neural network)
- Testing and validation (Training & Testing)

Palabras clave

- Aprendizaje Automático
- Vehículo autónomo
- Unity
- Python
- Redes Neuronales
- Procesamiento de imágenes
- Entrenamiento

Keywords

- Machine Learning
- Autonomous vehicle
- Unity
- Python
- Neural Networks
- Image processing
- Training

Índice

Índice de figuras

Capítulo 1

Introducción

Resumen: en este capítulo se expone la motivación que nos ha llevado a desarrollar este proyecto, así como los objetivos iniciales del mismo.

1.1. Motivación

Desde la creación del primer automóvil con motor de gasolina a finales del siglo XIX , el Benz Patent-Motorwagen, desarrollado y construido por el ingeniero alemán Karl Benz, el automóvil es uno de los productos de gran consumo que ha sufrido las evoluciones más notables[1].

Todo este proceso de evolución ha buscado en la mayoría de los casos ,una mejora de la conducción , la seguridad y el confort , por lo que no es de extrañar que a medida que avanzaban las tecnologías se buscase crear un vehículo capaz de conducirse a sí mismo, capaz de evitar los peligros a la conducción causados por culpa del propio conductor y permitiendo a sus ocupantes realizar otras actividades durante los trayectos.

Para lograr esto, el vehículo debe estar controlado por un "cerebro" software, capaz de analizar las diferentes situaciones que se puedan dar en el entorno . Cuando pensamos en el sistema de control de un vehículo autónomo es fácil caer en que sigue las normas clásicas de la programación del estilo " if..then..else ", por ejemplo : "Si un coche intenta adelantarte , aumentaras la distancia de seguridad y reducirás la velocidad facilitando la maniobra". No obstante , evidentemente uno se puede dar cuenta de que esta forma de idear el sistema es errónea , puesto que existen demasiadas posibles situaciones y no podrían controlarse todas.

Es aquí donde entra en uso de aprendizaje automático, que permite "entrenar " al sistema mediante el procesamiento de imágenes de las situaciones que van sucediendo. Al principio la tasa de errores del sistema será muy alta , pero a medida que se le vaya entrenando con más casos , será capaz de adaptar sus parámetros internos y tomar mejores decisiones.

1.2. Objetivos

El objetivo principal de este TFG es llevar a cabo una simulación en 3D de conducción autónoma , en la que el usuario entrene al vehículo realizando el recorrido por un circuito virtual , y con los datos obtenidos se entrenase un modelo mediante diferentes métodos de machine learning.

Para llevar a cabo los algoritmos de machine learning , se usará TensorFlow, una biblioteca de código abierto desarrollada por Google para construir y entrenar sistemas de redes neuronales. Otro requisito que deberá cumplir el sistema es que siga una arquitectura cliente-servidor entre la recogido de los datos y su procesamiento.

Las tareas que abordaremos podría clasificarse dentro de los siguientes campos de la informática:

- **Procesamiento de imágenes:** el sistema tendrá que aplicar técnicas de procesamiento sobre las imágenes recogidas por las cámaras del vehículo virtual.
- **Diseño de interfaces:** el simulador 3D que formará parte del sistema deberá tener una interfaz interactiva con el usuario.
- **Diseño de videojuegos:** a la hora de desarrollar el simulador se habrá de hacer uso de un motor de videojuegos 3D y seguir el proceso de desarrollo de un videojuego en el mismo .
- **Inteligencia artificial:** será necesario emplear redes neuronales para lograr un vehículo autónomo.

1.3. Visión general del documento

El documento está dividido en varios apartados que se indicarán a continuación:

- Capítulo 2: se explica el trabajo de investigación realizado antes de la implementación del sistema.
- Capítulo 3: se explica como se ha realizado la gestión del proyecto y las herramientas utilizadas para ello.
- Capítulo 4: se desarrolla el funcionamiento general de la aplicación, y se explican los principales componentes del sistema.
- Capítulo 5: se muestran los resultados en forma de tablas de una serie de pruebas seleccionadas de entre las realizadas.
- Capítulo 6: se expresan las conclusiones alcanzadas tras el desarrollo del proyecto y un posible trabajo futuro.
- Capítulo 7: se explican las contribuciones realizadas por cada miembro del equipo durante el proyecto.
- Apéndice A: se expone detalladamente los pasos seguidos durante el uso del sistema mediante un ejemplo guiado
- Apéndice B: se explican de forma sencilla los pasos para la instalación y prueba del sistema.

1.4. Motivation

Since the creation of the first gasoline-powered car in the late nineteenth century, the Benz Patent-Motorwagen, developed and built by the German engineer Karl Benz, the car is one of the consumer products that has undergone the most remarkable developments [1].

All this process of evolution has sought in most cases, an improvement in driving, safety and comfort, so it is not surprising that as the technologies advanced it was sought to create a vehicle capable of driving itself , capable of avoiding driving hazards caused by the driver's own fault and allowing his occupants to perform other activities during the journeys.

To achieve this, the vehicle must be controlled by a "brain" made of software, capable of analyzing the different situations that may occur in the environment. When we think of the control system of an autonomous vehicle it is easy to fall into the classic rules of the "if..then..else" style programming, for example: "If a car tries to overtake you, you will increase the safety distance and you will reduce the speed by facilitating the maneuver. " However, obviously one can realize that this way of devising the system is wrong, since there are too many possible situations and all could not be controlled.

This is where machine learning comes into use, which allows the system to be "trained" by processing images of the situations that are happening. At the beginning the error rate of the system will be very high, but as you train with more cases, you will be able to adapt your internal parameters and make better decisions.

1.5. Objectives

The main objective of this TFG is to carry out a 3D simulation of autonomous driving, in which the user trains the vehicle by going through a virtual circuit, and with the data obtained, a model will be trained through different machine learning methods.

To carry out machine learning algorithms, we will have the help of TensorFlow, an open source library developed by Google to build and train neural network systems. Another requirement that the system must comply with is that it follows a client-server architecture between the collection of the data and its processing.

The tasks we will be facing can be classified in the following fields of computer science:

- **Image processing:** The system will have to apply processing techniques on the images collected by the virtual vehicle cameras.

- **Interface design:** the 3D simulator that will be part of the system must have an interactive interface with the user.
- **Videogame design:** when developing the simulator, it will be necessary to use a 3D video game engine and follow the process of developing a videogame in it.
- **Artificial intelligence:** it will be necessary to use neural networks to achieve an autonomous vehicle.

1.6. General vision of the document

The document is divided in the following sections:

- **Chapter 2:** the research work carried out before the implementation of the system is explained here.
- **Chapter 3:** explains how the project management and the tools used for it have been carried out.
- **Chapter 4:** The results obtained in some of the tests performed are shown .
- **Chapter 5:** explains how the general operation of the application is developed, and the main components of the system.
- **Chapter 6:** shows the conclusions reached after the development of the project and the possible future work.
- **Chapter 7:** explains the contributions made by each team member during the project.
- **Appendix A:** the steps followed during the use of the system are explained in detail by means of a guided example.
- **Appendix B:** shows the steps for installing and testing the system in a simple way.

Capítulo 2

Trabajo previo

Resumen: en este capítulo se expone la historia y el avance de los vehículos autónomos y las tecnologías investigadas para desarrollar el proyecto.

2.1. Estado del arte

Los vehículos autónomos han tenido un avance considerablemente rápido en los últimos años , no obstante su historia comenzó mucho antes. Apenas años después del desarrollo del primer automóvil , se empezó a soñar con el desarrollo de vehículos sin conductor , fue así como en los años 20 se creó un vehículo controlado por radio capaz de encender el motor , cambiar de marcha y conducir sin nadie al volante[2].

Pasaron los años sin avances en este campo , hasta que en 1939 , en la exposición Futurama, Norman Geddes presenta un prototipo de vehículo eléctrico controlado por un circuito electrónico embebido en el pavimento de una carretera[3].

No fue hasta décadas después, en 1969 cuando se empezó describir ideas similares a lo que se concibe como vehículo autónomo en la actualidad. En ese año John McCarthy, uno de los padres de la inteligencia artificial , describió en un ensayo[4] lo que él denominó "chofer automático", un vehículo capaz de circular gracias a una entrada de cámara de televisión, haciendo uso de esta forma de la misma entrada visual de los seres humanos , la vista. McCarthy describió además que el vehículo debería recibir una localización introducida por el usuario y dirigirse inmediatamente hacia allí.

Por los años 80 , el profesor y experto en inteligencia artificial Ernst Dickmanns junto con su equipo de la Universidad de Munich, construyeron el primer vehículo realmente robotizado hasta el momento[5].
Una furgoneta Mercedes equipada con cámaras y sensores, modificada para controlar volante, acelerador y freno. El software del sistema era capaz de detectar en gran medidas las líneas de la carretera mediante el procesamiento de las imágenes detectando diferencias de colores, y mediante cálculos probabilísticos y computación paralela obtenían las respuestas.

Estas investigaciones fueron apoyadas por la Comisión Europea mediante el programa Eureka-Prometheus , logrando en 1994 que un Mercedes 500 SEL adaptado por el equipo de Dickmanns fuera capaz de recorrer 1000 km por las autovías de París. Un año después , el mismo equipo logró realizar un viaje Munich-Copenhague usando visión computarizada y un ordenador en tiempo real.

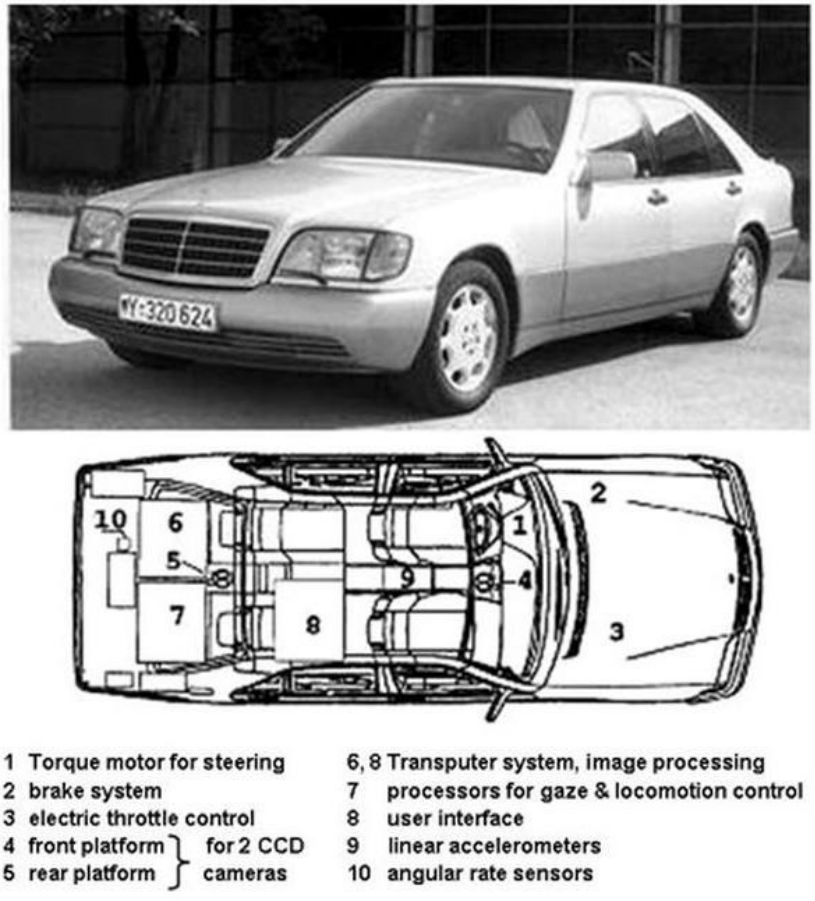

Figura 2.1: Mercedes 500 SEL usado por el equipo de Dickmanns.

También a principios de los años 90 , el investigador Dean Pomerleau, realiza una tesis doctoral sobre un estudio para demostrar cómo la aplicación de algoritmos basados en sistemas de redes neuronales , permitirá tomar imágenes en bruto de la carretera y enviar información a los controles en tiempo real. En 1995 , Pomerleau junto con su compañero Todd Jochem ponen a prueba su sistema de autoconducción "Navlab" en carreteras reales. Controlaban la velocidad y el freno, dejando el manejo de la dirección en manos de Navlab , llevando a cabo un viaje a través de Estados Unidos denominado "No hands across America"(Sin manos a través de América)[6].

Durante los años siguientes se llevaron a cabo algunos avances , remarcando los obtenidos gracias a los desafíos organizados por DARPA a principios de los años 2000. Como resultado de estos desafíos, destaca el proyecto ganador de 2005 de la Universidad de Stanford , cuyos miembros liderados por el ingeniero alemán Sebastian Thrun acabarían desarrollando el vehículo autónomo de Google , el Waymo. En 2015 una flota de este prototipo fue probada en el tráfico real en California dando grandes resultados.

En la actualidad muchas marcas de vehículos se han sumado a la era de los coches autónomos y han empezado a comerciar algunos modelos, destacando los modelos fabricados por Tesla , Audi o Mercedes entre otros.

Aunque el coche autónomo aun no esta perfeccionado del todo, esta muy claro que ya se ha convertido en una realidad , y no pasará mucho tiempo hasta que lo normal en nuestra sociedad sea tener uno .

2.2. Tecnologías relevantes y otras investigaciones

Para implementar el sistema ha sido necesario investigar distintas tecnologías que pudiesen ser de utilidad. En este apartado describimos las tecnologías estudiadas, y otras investigaciones, realizadas con el fin de cumplir con los objetivos del proyecto.

2.3. Python

Python[7] es un lenguaje de programación de alto nivel creado por Guido van Rossum, con una gran versatilidad , permitiéndole ser multiplataforma y multiparadigma. Python enfatiza la legibilidad del código y su limpieza , lo que facilita su utilización..

Es un lenguaje interpretado que presenta un fuerte tipado dinámico y una administración de la memoria automática , siendo además , compatible con variados paradigmas de programación , incluidos los imperativos , los orientados a objetos y también , aunque en menor medida , los funcionales.

Python contiene una librería estándar grande y muy completa , pero destaca aún más por la cantidad de librerías y documentación que tiene gracias a grupos de terceros.

En la actualidad uno de los usos más importantes de Python es el análisis de datos , por lo que la gran mayoría de las bibliotecas citadas anteriormente van dirigidas a ser usadas tanto en aprendizaje automático , como en la ciencia de datos , siendo esta la razón principal por la que hemos optado por usarlo.

La sección del proyecto dedicada a la codificación de algoritmos de aprendizaje automático, se ha sustentado a partir de ficheros .py, que constituyen los archivos propios de este lenguaje. Además también hemos hecho uso de notebooks que nos proporcionan un entorno de computación interactivo , en el que podemos combinar la ejecución del código de una forma muy visual a la vez que podemos comprobar los resultados de su testeo.

2.4. Motor de videojuegos Unity

Unity[8] es un motor de videojuegos lanzado en 2005 por la compañia Unity Technologies. Está programado esencialmente en C,C++ y C# y aunque se encuentra únicamente disponible en Windows, MAC OS y Linux , permite soporte de compilación con una gran variedad de plataformas objetivo.

Desde su salida en 2005 , Unity , ha revolucionado el mercado de los motores de videojuegos , irrumpiendo con fuerza por delante de otros competidores más asentados en el sector, como Unreal o CryEngine. Esto se ha debido en gran medida a la sencilla curva de aprendizaje y a la variedad de funcionalidades que presenta Unity en comparación con las alternativas, sin embargo , la circunstancia más determinante es que Unity es gratuito .

Sus creadores enfocaron el motor en cubrir las necesidades de aquellos desarrolladores de videojuegos que no tuvieran las capacidades ni el dinero necesario , para o bien crear su propio motor de videojuegos , o bien adquirir la licencia de uno ya existente . Siguiendo esta filosofía , Unity tiene a disposición de los desarrolladores una versión gratuita denominada personal, y dos versiones de pago , Unity plus y Unity Pro , no obstante la versión gratuita cuenta con las mismas características que las de pago a excepción de una restricción al compilar que obliga a mostrar el logo de Unity .

Figura 2.2: Logo de Unity.

Uno de los aspectos más destacados por el que decidimos usar Unity , es la gran cantidad de contenido descargable gratuito que existe actualmente en su asset store , tanto el creado de forma oficial por Unity Technologies, como el compartido de forma gratuito por otros usuarios. Esto paliaba en gran medida nuestra dificultad a la hora de diseñar elementos en 3D.

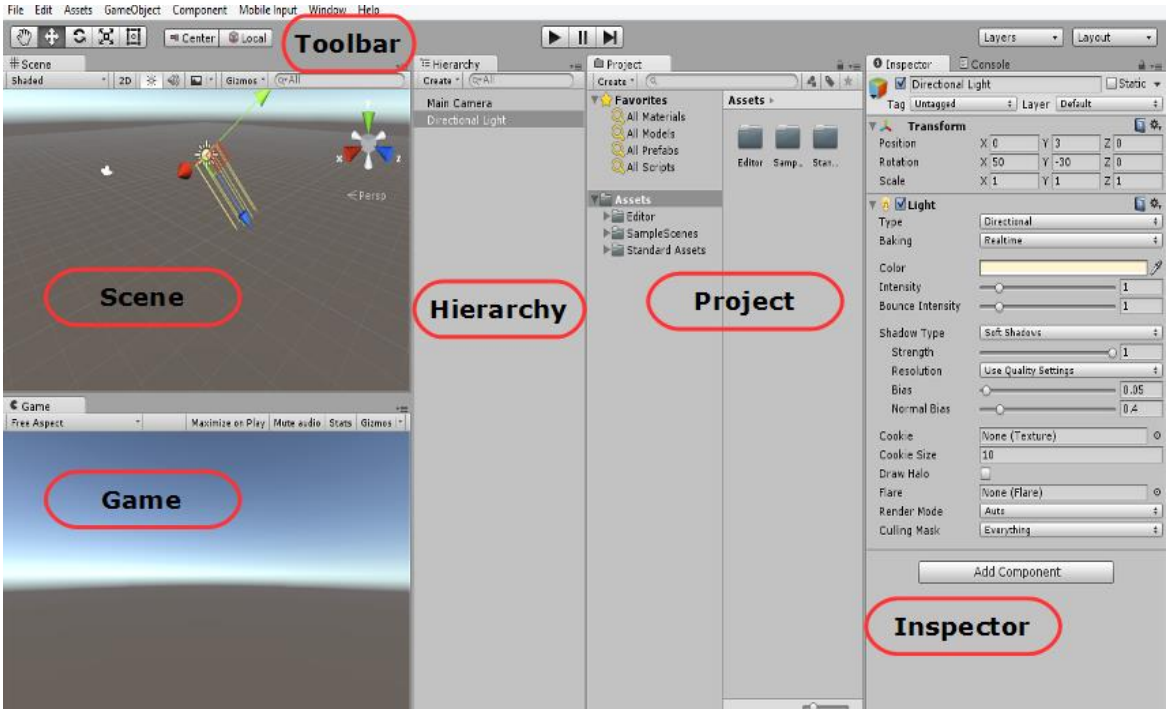

Figura 2.3: Vista del editor en Unity.

El editor de Unity está compuesto por diferentes ventanas unidas en una sola vista , de entre las que destacan 6 ventanas principales :

-Game: muestra la representación finalizada del juego renderizada por la cámara del juego, es una vista fija desde la que los elementos que la componen no pueden ser editados.

-Scene: muestra la escena , es decir , una de los escenarios del juego , a diferencia de Game , desde la vista de Scene sí pueden modificarse los elementos y entornos que la componen.

-Hierarchy: muestra cada elemento contenido en la escena actual, permite arrastrar un elemento sobre otro creando relaciones parentales , de forma que un objeto dependa de otro .

-Project: desde esta vista se puede acceder y gestionar todos los assets pertenecientes a nuestro proyecto , nos permite crear nuevos assets o eliminarlos.Se encuentra organizado mediante una estructura de carpetas y permite ver la jerarquía de las mismas en su panel izquierdo.

-Inspector: desde esta vista se pueden editar las propiedades o preferencias relacionadas con un GameObject concreto.Nos permite añadirle diferentes componentes a dicho objeto , como por ejemplo scripts o detectores de colisión.

-Toolbar: desde esta vista tenemos acceso a algunas de las funciones más esenciales de Unity. A la izquierda tenemos las herramientas básicas para manipular una escena y los objetos que en ella se sitúan , en el centro se encuentran los controles de reproducción del juego , y en la zona derecha tenemos acceso a los menús de visibilidad de las capas en las que se divide la escena.

Figura 2.4: Vista detallada de la Toolbar de Unity

Actualmente Unity soporta tres tipos de lenguaje:

-UnityScript: lenguaje propio de Unity , aunque a la hora de la verdad es muy similar a Javascript.

-C#: lenguaje muy similar a java, estandarizado por Microsoft con su tecnología .NET.También bebe de otros lenguajes como C++.

-Boo: lenguaje con una sintaxis similar a Python .

Finalmente decidimos que Unity seria la mejor opción a la hora de desarrollar nuestro proyecto a raíz de estas razones:

-**No presenta una curva de aprendizaje pronunciada** , por lo que no requiere tanto tiempo de aprendizaje y adaptación como presentan otros motores de videojuegos.

-**La assets store de Unity** nos permite tener acceso a diseños o templates de juegos de forma gratuita, cubrienda ciertas carencias a la hora de diseñar por nosotros mismos .

-**La gran cantidad de información disponible** , tanto en forma de documentación como de tutoriales.

2.5. C sharp - C#

Se trata de un lenguaje de programación orientado a objetos , que ha cobrado gran importancia al ser estandarizado por Microsoft en su plataforma .NET.

Su sintaxis deriva de C++ e incluye un modelo de objetos muy similar al también utilizado en Java.

C# fue creado en 1999 por Anders Hejlsberg y su equipo de ingenieros en Microsoft, que lo llamaron Cool en un comienzo pero por problemas de licencias lo llamaron C#, siendo "#" el equivalente a cuatro símbolos "+".

Este lenguaje es uno de los soportados para el desarrollo de scripts en Unity , por lo que debido a la experiencia previa que teníamos en su uso hemos decidido utilizarlo en nuestros propósitos.

2.6. TensorFlow

Es una librería de código abierto lanzada por Google en 2015 que proporciona una interfaz para expresar algoritmos de aprendizaje automático y su implementación. Fue creado con el objeto de satisfacer la necesidad de construir y entrenar redes neuronales en los sistemas de Google.

Tiene una arquitectura en forma de grafo que hace posible su ejecución en tanto en CPU's como en GPUs , así como en otros dispositivos.

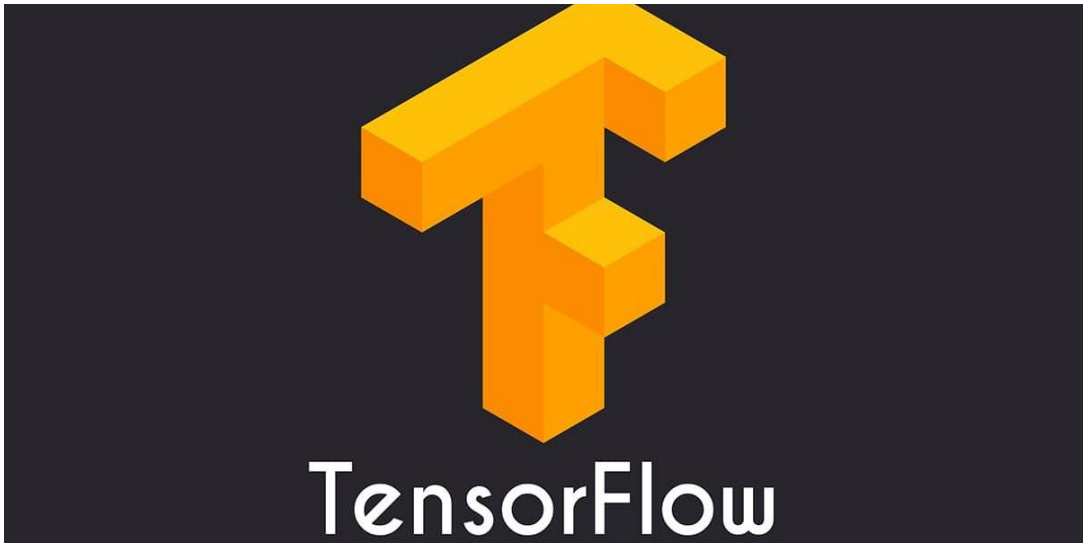

Figura 2.5: Logo de TensorFlow

Al tratarse de un sistema flexible puede ser usado en una gran diversidad de algoritmos diferentes y por esta razón ha sido utilizado en muchos campos de la informática[9] , como reconocimiento por voz , visión computacional , procesamiento del lenguaje, etc.

TensorFlow utiliza un solo flujo de datos a la hora de representar todos los cálculos y los estados en un algoritmo de aprendizaje automático , incluyendo las operaciones matemáticas y los parámetros. Este tipo de cálculo en forma de flujo facilita la ejecución de cálculos independientes de forma paralela y el reparto de la carga de computación entre varios dispositivos.

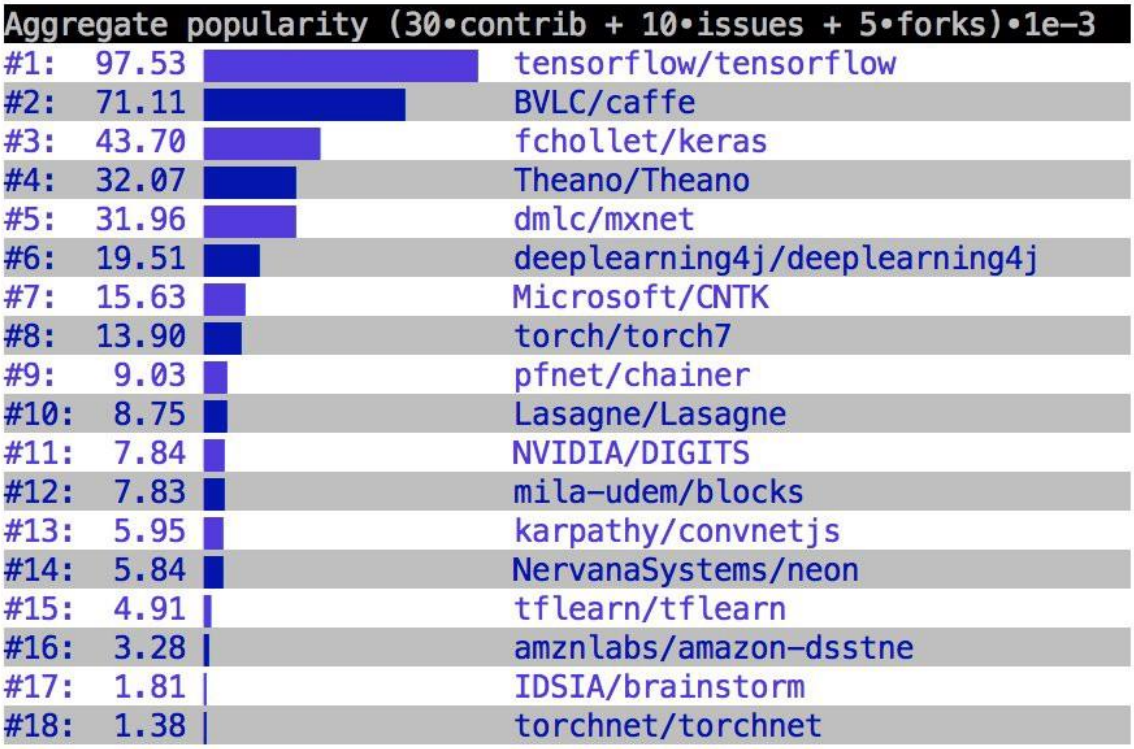

Figura 2.6: Popularidad librerías de aprendizaje automático

2.7. OpenCV

Se trata de una conocida librería de código abierto destinada a labores de visión computarizada desarrollada por Intel y lanzada en el año 2000. Fue construida para proporcionar un entorno de desarrollo sencillo y fácil de utilizar a la hora de crear una infraestructura común para aplicaciones de visión computarizada.

La librería permite trabajar con una gran cantidad de algoritmos de visión artificial y de machine learning, que pueden ser utilizados para reconocimiento facial , rastrear y detectar objetos , edicion de imagenes y de video y otros muchos usos.

Grandes compañías hacen uso de esta herramienta a la hora de crear sistemas de detección por videovigilancia o software de identificación de objetos para robots.

Proporciona interfaces en C++, Java , Matlab y Python , razón por la cual , unida al hecho de que ya conocíamos la existencia de esta herramienta , decidimos optar por su uso en el proyecto a la hora de procesar nuestras imágenes.

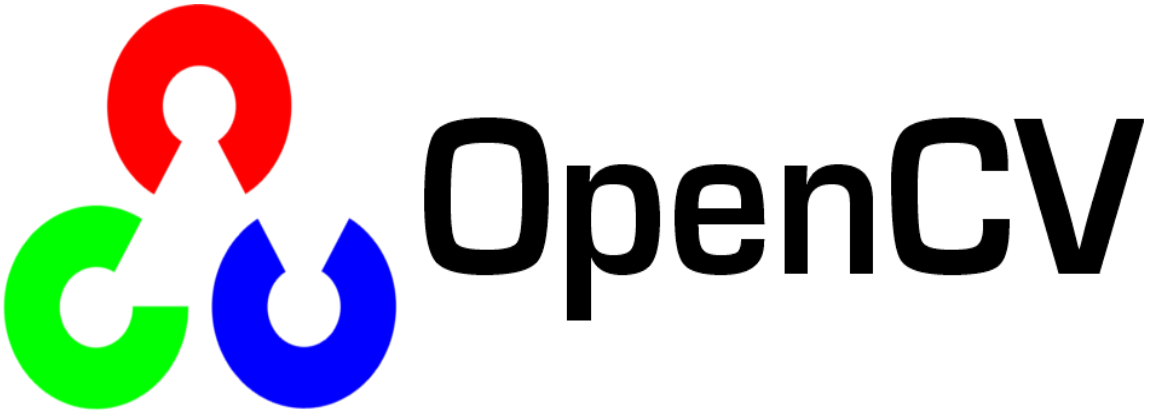

Figura 2.7: Logo de OpenCV

2.8. Numpy

Numpy[10] es una librería opensource para Python que le aporta mayor soporte a la hora de realizar calculos cientificos con vectores y matrices.

Fue creada por Travis Oliphant en 2005 , basándose en Numeric , una librería anterior creada por Jim Hugunin diez años antes. Oliphant hizo un codigo mas facil de mantener y lo suficientemente óptimo y flexible para implementar las características novedosas de manejo de arrays que implementa Numarray.

La importancia de Numpy ante todo es su capacidad de proporcionar estructuras para encapsular diferentes tipos de datos implementados de manera propia, como arrays o matrices, las cuales son mucho más rápidas y eficientes que las proporcionadas de forma estándar por Python.

2.9. Scipy

Scipy[10] es una colección de algoritmos matemáticos y funciones de código abierto, nacida a raíz de la colección de algoritmos creada por Travis Oliphant en 1995 denominada Multipack. Se basa en la extensión Numpy de Python .

Proporciona a los usuarios comandos y clases de alto nivel para visualizar y manipular datos.. El uso de Scipy con Python crea una interfaz de procesamiento de datos que rivaliza con herramientas reconocidas como Octave o Matlab. La gran cantidad de operaciones de optimización y su compatibilidad con herramientas como Numpy la hace idónea para nuestros propósitos en el aprendizaje automático.

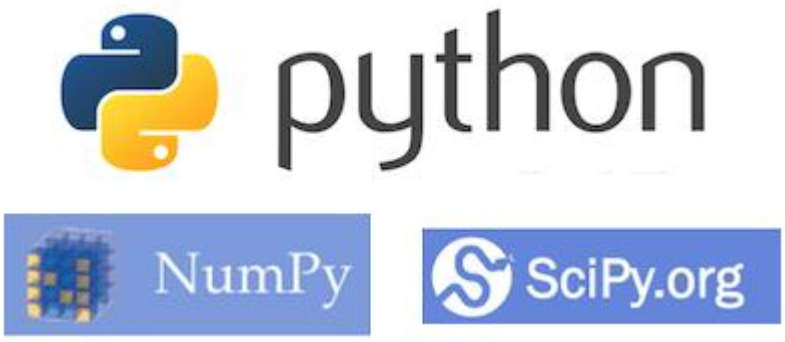

Figura 2.8: Logos de Numpy y Scipy en Python

2.10. Scikit-Learn

Scikit-Learn es una librería de aprendizaje automático de código libre para programar en Python.

Comenzó como un proyecto de David Cournapeau durante el "Google Summer of Code" de 2007 . Pero no fue hasta 2010 cuando gracias al esfuerzo del Instituto Francés en Computación y Automatización (INRIA) se hizo su primer lanzamiento público.

Está diseñado para interoperar con las bibliotecas de Numpy y Scipy y cuenta con algoritmos para resolución de problemas de clasificación , regresión , clustering o preprocesado.

Actualmente se encuentra mantenida por voluntarios y sigue en desarrollo activo.

2.11. Scikit-Image

Scikit-Image es una librería de procesamiento de imágenes de código libre para programar en Python.

Su versión inicial fue lanzada en 2009 por Stefan van der Walt, como una extensión para el tratamiento de imágenes para Scipy.

Contiene algoritmos para manipulación del color o exposición , para transformaciones geométricas , detección de bordes o aplicación de filtros entre otros.

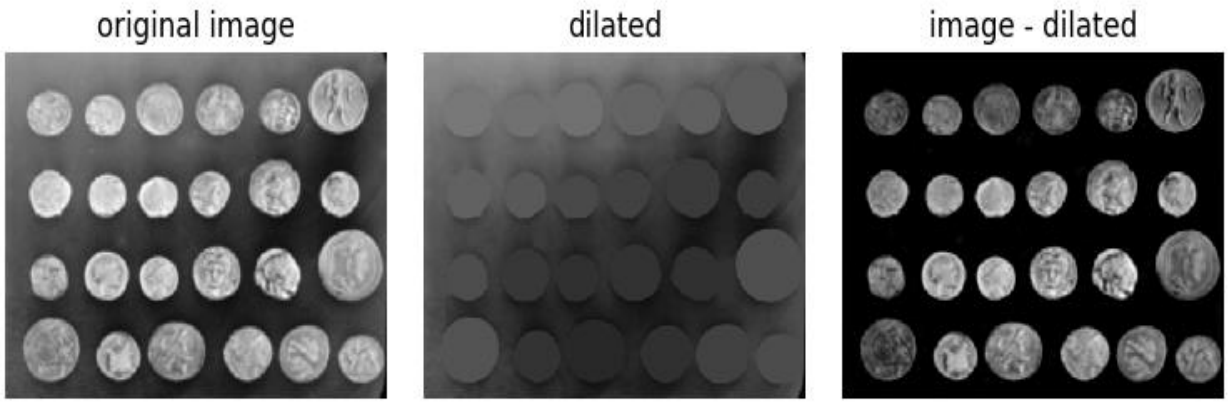

Figura 2.9: Ejemplo de tratamiento de imágenes con Scikit-Image

2.12. Keras

Keras es una API de código abierto , para trabajar con redes neuronales de alto nivel , escrita en Python y capaz de ejecutarse sobre TensorFlow.

El proceso de desarrollo de Keras se llevó a cabo en 2015 durante el proyecto ONEIROS(Sistema Operativo de Robot Inteligente Neuroelectrónico de código abierto) y su principal desarrollador fue el ingeniero de Google François Chollet.

Keras no presenta una curva de aprendizaje pronunciada lo que explica porque permite crear prototipos de forma sencilla y rápida.
Además permite trabajar con redes convolucionales y está diseñado para poder ejecutar los cálculos tanto en CPU's como en GPU's , lo cual toma una gran importancia a la hora de maximizar rendimiento que requerimos en nuestro proyecto.

Actualmente Keras cuenta con más de 200000 usuarios , convirtiéndola en unas de las herramientas favoritas entre los investigadores de aprendizaje profundo, haciendo que haya sido adoptada por investigadores de reconocidas organizaciones científicas como la NASA o el CERN[11].

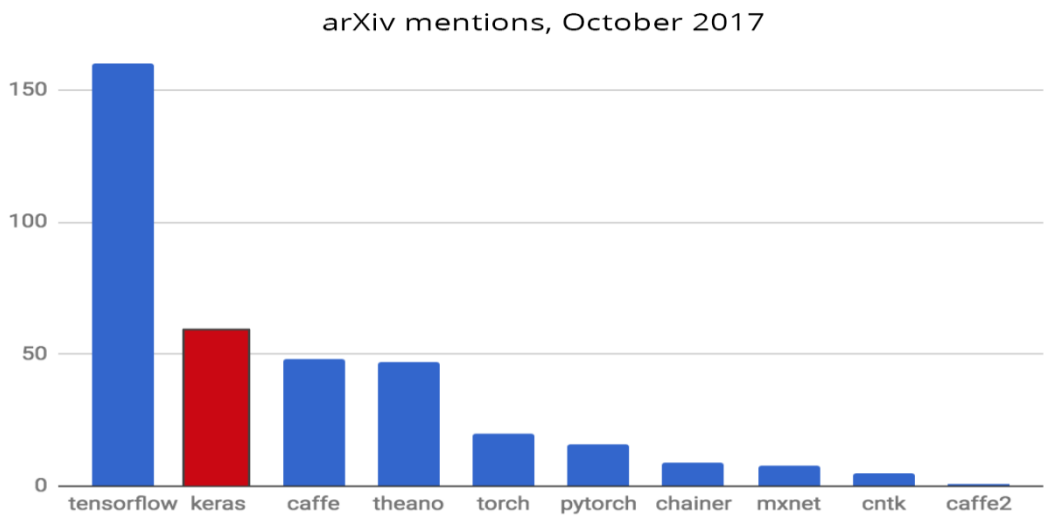

Figura 2.10: Menciones de Keras en artículos científicos

2.13. Pandas

Pandas[12] es una librería de código libre desarrollada por Wes McKinney, fue diseñadas para funcionar como una extensión de Numpy facilitando de gran manera los trabajos a la hora de manipular grandes datasets.

Este paquete para Python proporciona estructuras de datos rápidas, flexibles y diseñadas para ser intuitivas y fáciles de usar.

Actualmente el equipo que mantiene esta librería tiene el objetivo de convertirla en la herramienta de análisis y tratamiento de datos más potente de código libre , y ya están en camino hacia ese objetivo.

Algunas de sus funcionalidades más destacables son:

-Mutabilidad de tamaño: las columnas se pueden insertar y eliminar del DataFrame.
-Reestructuración y segmentación de conjuntos de datos.
-Conjuntos intuitivos de unión de datos.
-Herramientas de entrada/salida robustas que permiten cargar datos desde archivos planos(CSV), archivos de excel , bases de datos…
-Alineación de datos de forma automática.

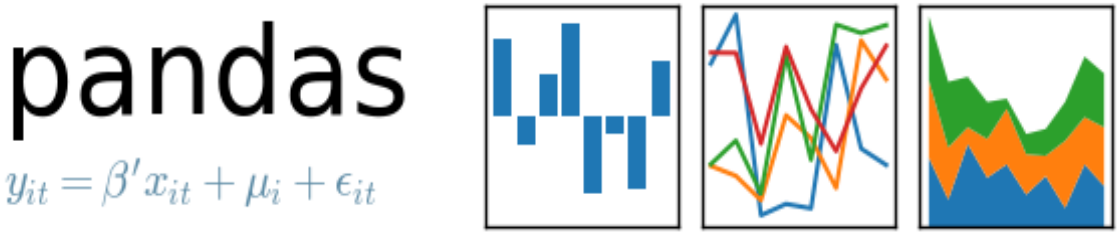

Figura 2.11: Logo de Pandas

2.14. H5py

H5py[13] es el nombre que recibe una librería de código abierto para python que hace posible trabajar con el formato de datos HDF5.

HDF5 hace referencia a "Hierarchical Data Format 5" , un tipo de formato de archivo diseñado para almacenar y manipular grandes cantidades de datos. Esto nos permite organizar grandes cantidades de datos de carácter numérico y trabajar con ellos de forma sencilla desde Numpy.

H5py nos proporciona una forma óptima de almacenar y trabajar con las matrices de pesos que componen el modelo de una red neuronal.

2.15. Pillow

Pillow es una biblioteca de código abierto para Python que añade funcionalidades la hora de trabajar , abrir o guardar diferentes formatos de imagen.
Su primera versión data de 1995 , época en que Fredrik Lundh la desarrolló usando Python y C.

Pillow ofrece una gran cantidad de procedimientos estándar para la manipulación de imágenes , tales como recortes , cambios de tamaña, rotaciones , filtros de escala de grises , saturación , etc.

En Pillow todo funciona en torno a una clase , "Image", proporciona un objeto imagen sobre el que se realizan la manipulaciones, y permite instanciar esta clase cargando de archivo , como resultado de manipular otras imágenes o desde cero.

```
1 from PIL import Image, ImageFilter  # imports the library
2
3 original = Image.open("file.ppm") # load an image from the hard drive
4 blurred = original.filter(ImageFilter.BLUR) # blur the image
5
6 original.show() # display both images
7 blurred.show()
```

Figura 2.12: Ejemplo de carga de imagen y desenfocado con Pillow

Capítulo 3

Metodología de trabajo

Resumen: en esta sección describimos las metodologías que hemos seguido al desarrollar el proyecto , así como las herramientas de las que hemos hecho uso para ayudarnos a la hora de gestionar el trabajo.

3.1. Introducción

Decidimos comenzar el proyecto realizando una división en lo que considerábamos las diferentes ramas del proyecto: investigación , simulación 3D y algoritmia.

Primeramente comenzamos con la parte de investigacion y recuperacion de informacion. La cual subdividimos entre herramientas de aprendizaje automático , herramientas de procesado de imágenes y herramientas para uso de Unity.

Buscamos la información en Internet, tanto en las páginas oficiales , como en otras bases de datos o la propia Wikipedia.En el caso de la sección del simulador de Unity se realizaban pruebas para comprobar que extensiones podrían resultar útiles. En la sección de algoritmia se llevaban a cabo pruebas de diferentes herramientas para comprobar cual podría adaptarse mejor a nuestras necesidades , tanto para comprobar sus funcionalidades como para estudiar su curva de aprendizaje.

[1]Más sobre Wikipedia: https://es.wikipedia.org/wiki/Wikipedia:Portada

En las siguientes dos fases del proyecto , pusimos en práctica los conocimientos adquiridos durante la etapa de investigación. En un primer momento optamos por que uno de nosotros se centrará en la creación del simulador y el otro en los scripts de algoritmos . No obstante a medida que avanzaba el desarrollo dejamos de cumplir de forma estricta este reparto de tareas y ambos trabajabamos en ambas secciones .

De cara a gestionar estos repartos de tareas y controlar las versiones del proyecto, empezamos a usar herramientas como Trello y Github, y como medio de almacenamiento auxiliar hicimos uso de Google Drive .

3.2. Trello

Trello es un software de gestión de proyectos con interfaz en web principalmente.

El uso de esta herramienta nos ha resultado muy útil de cara a gestionar el reparto de tareas en el proyecto.

Trello permite crear tableros , pudiendo así separar el reparto de las tareas para las diferentes ramas del proyecto. Nosotros decidimos dividir el proyecto entre los siguientes tableros:

- **Investigación**: en este tablero organizaremos el reparto de tareas a la hora de investigar una u otra herramienta que pueda resultar de interés para el proyecto.
- **Simulador de conducción**: en este tablero se repartían las tareas relacionadas con el diseño y desarrollo del simulador 3D del vehículo en Unity.
- **Cerebro del Vehículo**: En este tablero gestionamos las labores a realizar en lo respectivo a los scripts de entrenamiento y testeo de las redes neuronales.

Figura 3.1: Tableros propios usados en Trello

3.3. GitHub

Es una reconocida plataforma para alojar proyectos y gestionar sus diferentes versiones mediante Git.

Fue lanzada en 2008 por Ruby on Rails , aunque ha sido comprada por Microsoft en 2018 por varios miles de millones de dólares.

Permite trabajar de forma colaborativa entre programadores, facilitando las actualizaciones de código.

Las principales razones que nos han llevado a usarlo han sido entre otras: su popularidad actual , siendo de hecho una seña de identidad en muchos Currículums , las funcionalidades que ofrece , y especialmente que es la herramienta de control de versiones que hemos estado usando a lo largo de la carrera.

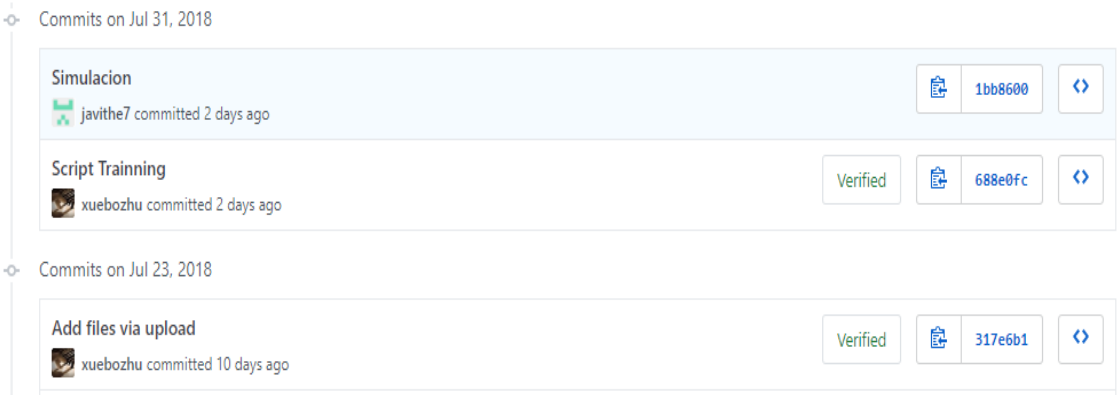

Figura 3.2: Ejemplo de commits en el proyecto

Aunque a la hora de la práctica , al ser un grupo de solo dos personas, y trabajar en paralelo en persona normalmente , realmente la mayoría de las subidas nuevas al proyecto solo las hacia uno de nosotros por lo que no aprovechamos esta vez todas las funcionalidades que nos ofrece github.

3.4. Google Drive

Aunque en un primer momento no consideramos necesario ningún tipo de almacenamiento de datos en internet debido a que trabajabamos en el proyecto en persona , finalmente consideramos que una buena forma de archivar las herramientas y datos considerados de interés sería esta.

Google Drive es un servicio de almacenamiento cuya versión tal y como la conocemos hoy fue lanzada por Google en 2012.

Actualmente permite un almacenamiento gratuito de 15 GB para cada uno de sus usuarios y permite además visualizar y editar documentos entre varios usuarios a la vez , lo cual nos ha resultado muy útil de cara a la toma de apuntes a modo de blog de notas.

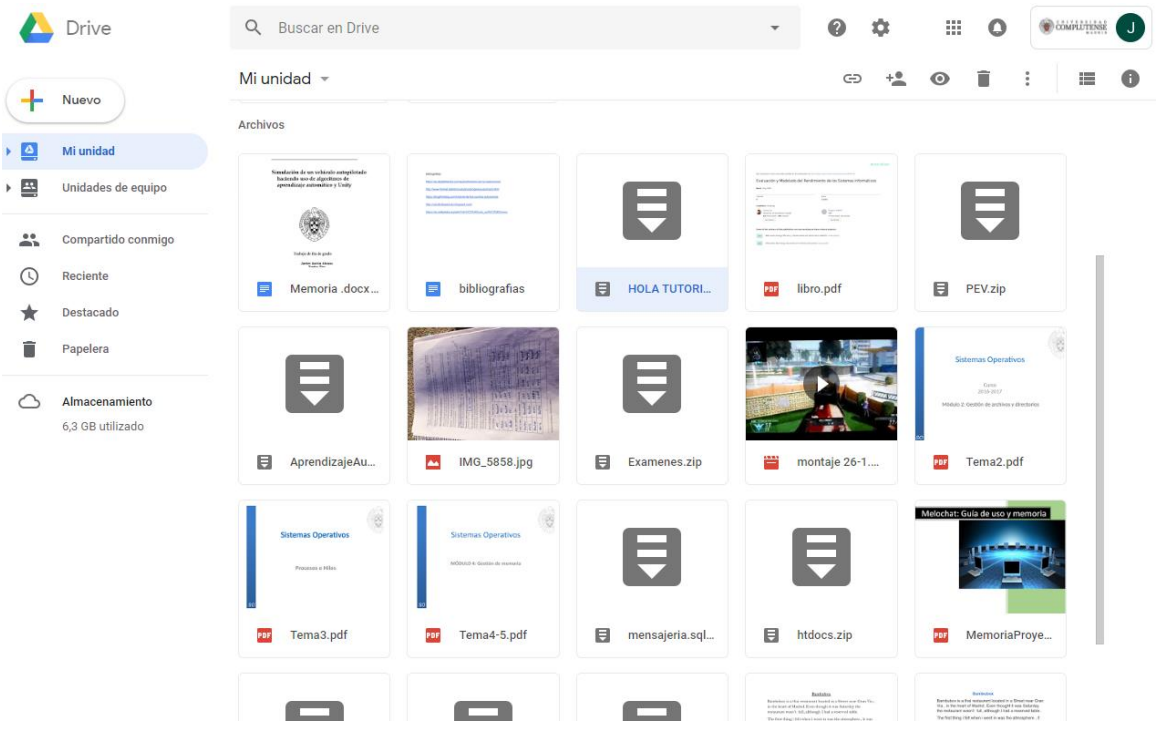

Figura 3.3: Ejemplo de vista de archivos en Drive

Capítulo 4

Simulador de autoconducción en Unity 3D

Resumen: se exponen de manera genérica las características del sistema, en nuestro caso, el simulador de autoconducción en Unity 3D.

4.1. Funcionamiento general del sistema

El sistema creado durante el proyecto está inspirado en los estudios llevados a cabo por la empresa NVIDIA en su escrito "End to End of autonomous driving"[15], donde exponen información muy importante de cara a crear un vehículo autopilotado a este nivel de abstracción , siendo principalmente nuestra fuente de conocimientos y ayuda para crear un sistema de esas características.

El funcionamiento del sistema se divide en dos fases , la fase de entrenamiento y la fase de conducción autónoma.

A continuación se exponen explicaciones del funcionamiento de cada una de las fases sin entrar en apartados técnicos.

4.1.1. Fase de entrenamiento

Durante la fase de entrenamiento sentamos las bases de aprendizaje del vehículo .

El usuario toma el control del vehículo virtual en un circuito cerrado dentro del simulador 3D. Dicho coche esta modificado para poder contar con los sensores necesarios para la recolecta de datos.

Cuenta con 3 cámaras de grabación de imágenes sobre el capó y su ángulo de giro del volante , velocidad y aceleración se encuentran monitorizados.

Figura 4.1: Vista del vehículo y de sus cámaras

El usuario es quien debe activar y desactivar la grabación del recorrido gracias a un botón en la interfaz de entrenamiento. Una vez activada la grabación y el usuario empiece a pilotar el vehículo por el circuito, el sistema almacenará en un archivo csv (ideal para almacenamiento de datos en forma de tabla) cada una de las muestras obtenidas en función de la frecuencia de barrido.

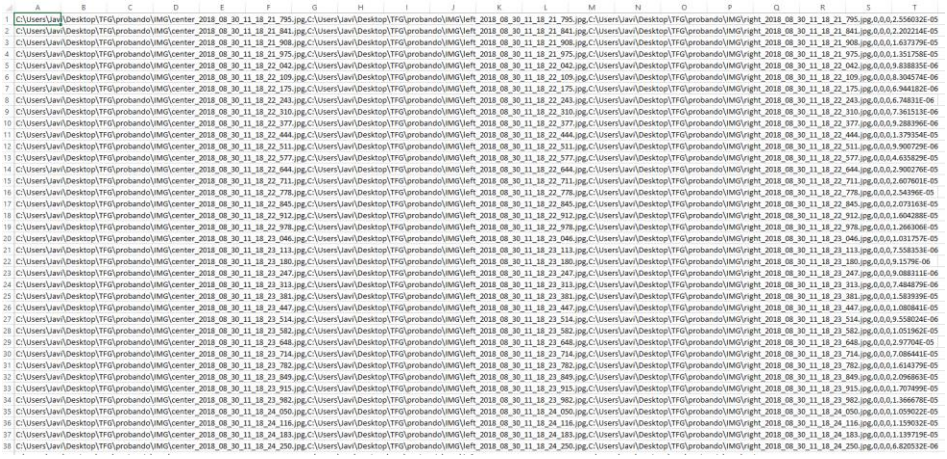

Figura 4.2: Captura del registro CSV con los datos obtenidos al entrenar.

Estos datos obtenidos son usados para entrenar un modelo de red neuronal convolucional construida en python siguiendo las explicaciones de NVIDIA. Este proceso de entrenamiento ajusta los pesos internos de la red , llevando a cabo de esta forma un proceso de aprendizaje, dándonos como resultado que la red haya “aprendido”.

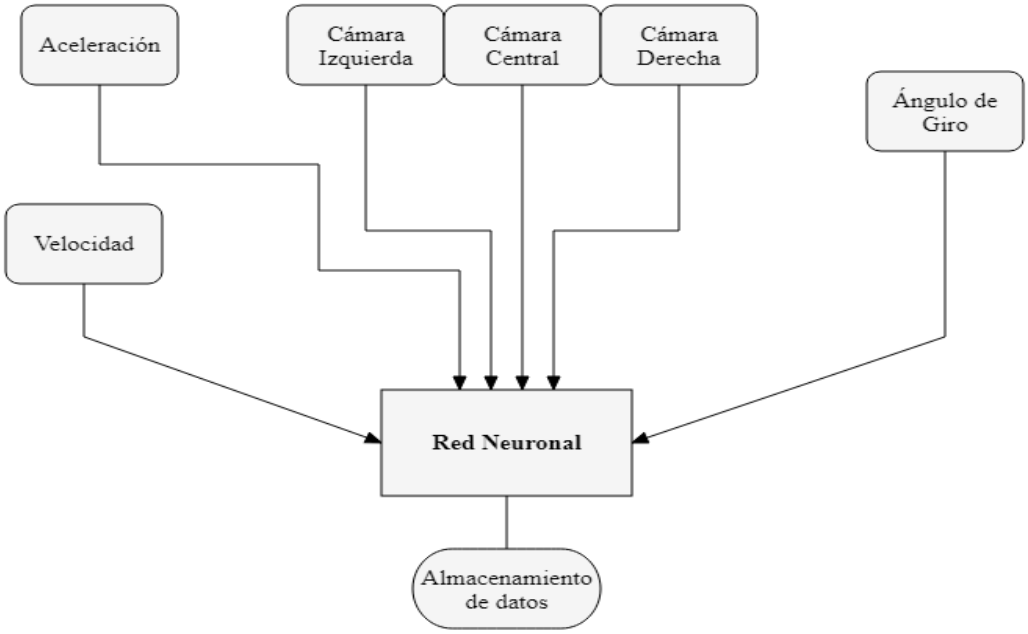

Figura 4.3: Diagrama que representa en alto nivel la recolección de datos en el proceso de entrenamiento.

4.1.2. Fase de conducción autónoma

Durante la fase de conducción autónoma ponemos a prueba lo "aprendido" por la red neuronal para poder conducir el vehículo por el circuito de forma autónoma y sin asistencia humana.

En esta fase diferenciamos dos elementos claves que trabajan en paralelo durante su funcionamiento siguiendo una arquitectura cliente-servidor: por un lado el simulador 3D que hace las veces de cliente y por otro el script de conducción en python , que hace las veces de servidor.

El simulador en la fase autónoma no dista mucho de la versión de entrenamiento , el vehículo es el mismo y el circuito también , pero en este caso el vehículo no puede ser controlado manualmente y solo cuenta con una cámara sobre su capó , la central , que es utilizada para mandar imágenes del recorrido a los scripts de conducción.

Cuenta con un controlador que recibe los resultados enviados desde el servidor python , y pasa esos datos al vehículo , que gira o acelera en consecuencia.

Por el otro lado se encuentran los scripts de conducción en python , haciendo las veces de servidor. Estos reciben la imagen desde el simulador y haciendo uso del modelo de la red neuronal entrenada previamente obtiene las predicciones , es decir la velocidad y giro de volante que debe llevar el vehículo dadas las características del tramo de carretera del que ha sido obtenida la foto.

Estas predicciones son enviadas en forma de paquete de datos al simulador , momento en el que estos datos son procesados para asignar la cantidad de impulso que hay que dar al giro o al acelerador .

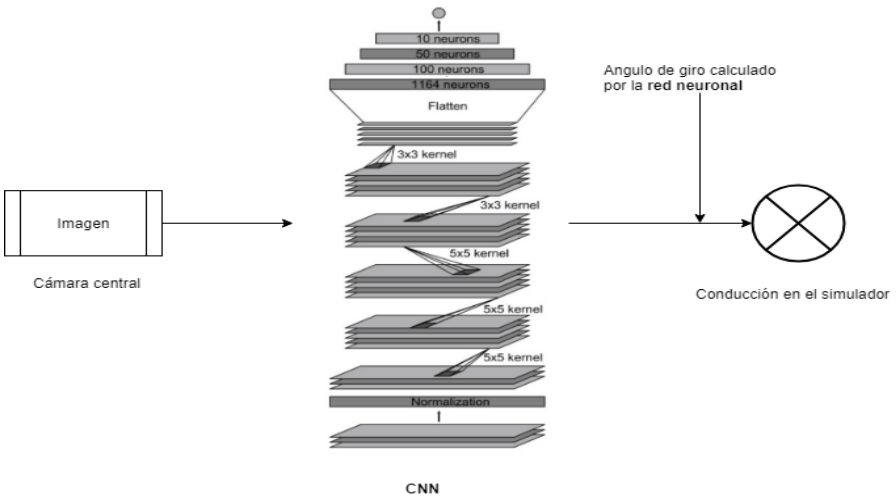

Figura 4.4: Ejemplo de conducción a partir de la red entrenada

4.2. Componentes del sistema

El sistema realizado en este proyecto consta de dos partes claramente diferencias en lo que a los elementos que lo componen se refiere . Por un lado tenemos la parte realizada en el motor de videojuegos Unity 3D , que se corresponde al simulador y es la parte con la que interactúa el usuario. Por otra parte tenemos la dedicada a los scripts en python , que son los encargados de utilizar los datos recolectados por el simulador y crear la red neuronal para posteriormente hacer las veces de servidor durante la conducción autónoma.

A continuación se exponen los componentes principales de cada una de estas dos partes

Figura 4.5: Ejemplo de vista de componentes del simulador en Unity

4.2.1. Componentes Simulador 3D

La elección de Unity como motor de videojuegos sobre el que desarrollar el simulador viene dada por varias razones , algunas ya explicadas anteriormente , pero la principal en nuestro caso fue la gran cantidad de paquetes que pueden ser descargados desde la Asset Store , tanto los creados oficialmente por Unity como los creados por otros usuarios.

Concretamente para la creación del simulador nosotros hicimos uso de 3 paquetes gratuitos de la Asset Store:

- **MS Vehicle System[16]:** este paquete gratuito contiene un sistema de vehículos 3D , tanto a nivel de diseño, como de sus correspondientes controladores de conducción manual.
- **Environmental Race Track Pack[17]:** este paquete también gratuito contiene 4 circuitos simples diseñados con pocos polígonos , lo que lo hace idóneo para simulaciones de conducción donde la calidad gráfica no es la primordial pero si el rendimiento , como es nuestro caso. Nosotros decidimos hacer uso del circuito "Coastal Track" para nuestras pruebas.
- **Socket.IO for Unity[18]:** se encarga del análisis sintáctico y semántico (entre otras funciones) de las sentencias en lenguaje natural, tanto las introducidas por el usuario como las obtenidas de las distintas fuentes de conocimiento utilizando el módulo de obtención de datos comentado anteriormente.

Tras cerciorarnos que estos paquetes no dieran conflictos entre sí y que habían sido cargado correctamente, empezamos a desarrollar el entorno del simulador. Para ello seguimos los pasos estándar para desarrollar proyectos en Unity , es decir , el uso de las escenas.

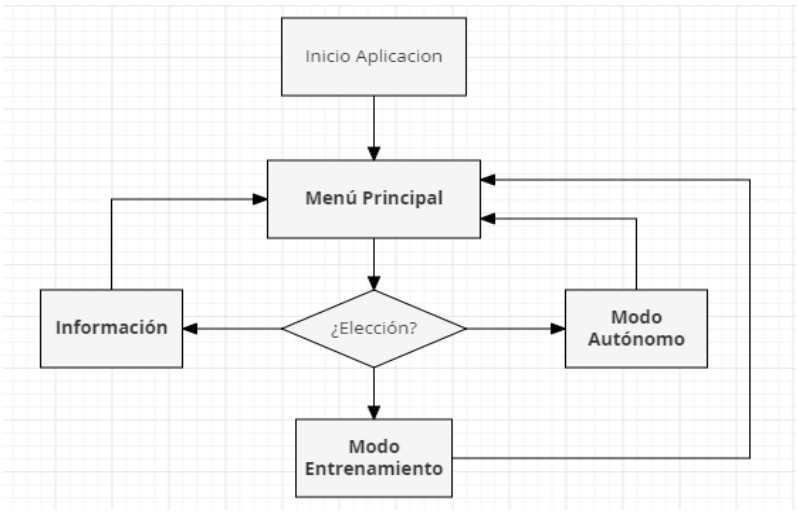

Figura 4.6: Diagrama de flujo entre escenas del simulador

4.2.1.1. Escena “Menú Principal”

Dentro de esta escena se muestra mediante una sencilla interfaz gráfica con botones , el menú central del simulador, en el cual nos aparecen 4 opciones : visualizar la información del sistema , acceder al modo de entrenamiento del vehículo , acceder al modo de conducción autónoma y cerrar la aplicación .

Figura 4.7: Escena Menú

4.2.1.2. Escena “Información”

Consistente de una sencilla interfaz gráfica con un solo botón , en esta escena se explica el funcionamiento del simulador y sus controles . En esta misma escena también se muestran la información relativa a los créditos del proyecto.

Figura 4.8: Escena Información

4.2.1.3. Escena "Modo Entrenamiento"

Nos encontramos en una de las escenas con más importancia del sistema , la encargada de recoger los datos producidos por la conducción manual del vehículo.

El objetivo que persigue esta escena consiste en conducir a lo largo de un circuito y recolectar datos del entorno.

Para recoger estos datos se han realizado una serie de modificaciones y extensiones en el script en C# "MSVehicleController" , este script estaba incluido en el paquete MS Vehicle System , e incluye todo el código base necesario para la conducción del vehículo y el funcionamiento de sus físicas de derrape o amortiguación entre otras.

Dentro de este script se han realizado las modificaciones necesarias para poder incluir las cámaras al vehículo , activación y desactivación de la grabación de datos , y el volcado de estos datos sobre un archivo de almacenamiento de tablas en formato .csv. Las imágenes por su parte son almacenadas en una carpeta IMG contigua al registro .csv.

Figura 4.9: Escena de entrenamiento al momento de activar la grabación

4.2.1.4. Escena "Modo Autónomo".

Esta escena es la encargada de mostrarnos al vehículo ya autonomo conducir por el circuito , es muy similar a la escena de entrenamiento , pero en esta el usuario no tiene ningún tipo de control sobre el vehículo.

Aunque el controlador del vehículo también se basa al del paquete MSVehicleSystem , las modificaciones realizadas sobre este han sido diferentes al del vehículo entrenable , por la principal naturaleza del mismo , este vehículo recibe datos , el otro los guarda. No obstante este vehículo debe enviar continuamente la imagen que le brinda su cámara central al servidor python.

De cara a intercambiar los datos con el servidor python usamos el script "CommandServer" , este script es el encargado de establecer el socket de conexión con el servidor python haciendo uso del paquete "SocketIO for Unity". Además de el establecimiento de la conexión, es en este script donde se encapsulan los datos enviados al servidor python (la imagen de la cámara central) y donde se recibe el paquete de datos con los comandos de movimiento que debe realizar el vehículo.

Figura 4.10: Escena de conducción autónoma

4.2.2. Componentes del servidor Python

Para la parte del Modo Autónomo será necesario primeramente entrenar un modelo para ello se utiliza una red neuronal para clonar el comportamiento de conducción de automóviles. Es un problema de regresión supervisada entre los ángulos de dirección del automóvil y las imágenes de la carretera frente a un automóvil.

La carpeta que contiene los scripts de Python consta de tres scripts principales, inspirados y adaptados de los trabajos de Naoki Shibuya[19]:

- **Script de construcción y entrenamiento del modelo (model.py):** En este script se utilizan varias librerías de manipulación y análisis de datos para poder construir y entrenar el modelo.
- **Script de tratamiento de imágenes (utils.py):** Este script contiene funciones de utilidad para el tratamiento de imágenes. Finalmente, genera la imagen de entrenamiento a la cual le asocia la ruta del fichero donde se encuentra en local y el ángulo de giro correcto.
- **Script de conducción (drive.py):** Por último en el script de conducción se crea el servidor Python que hará conexión vía websocket con el cliente, en este caso nuestro Simulador en Unity. El servidor mandará datos de giro y aceleración en todo momento al cliente para pilotar el vehículo acorde al resultado del entrenamiento, a partir del modelo entrenado con los pesos asignados.

4.2.2.1 Script de construcción y entrenamiento del modelo

Formado por el script **model.py** pretende crear y entrenar el modelo. De esta forma le asigna pesos a nuestra red neuronal convolucional sacada de NVidia específicamente para resolver este tipo de problemas de procesamiento de datos visuales para coches autónomos. Este script depende del script **utils.py** para el procesamiento y tratamiento de imágenes.

Dicho script está compuesto por las siguientes partes:

-*Carga de datos*: En general, esta carga los datos de entrenamiento y divide el conjunto en entrenamiento y validación. Para ello, lee el archivo CSV->driving_log.csv desde el directorio 'data'.

Columnas driving_log.csv:

1- Path Imagen Central

2- Path Imagen Izquierda

3- Path Imagen Derecha

4- Ángulo de Giro (-30 a 30) / Máximo ángulo de giro (30)

5- Aceleración (0 a 1)

6- Freno

7- Velocidad (0 Kmh a 30 Kmh(vel. Máxima))

-*Construcción del modelo*: Se centra en construir el modelo, en este caso la Red Neuronal Convolucional basada en el modelo de NVidia para la solución de este problema.

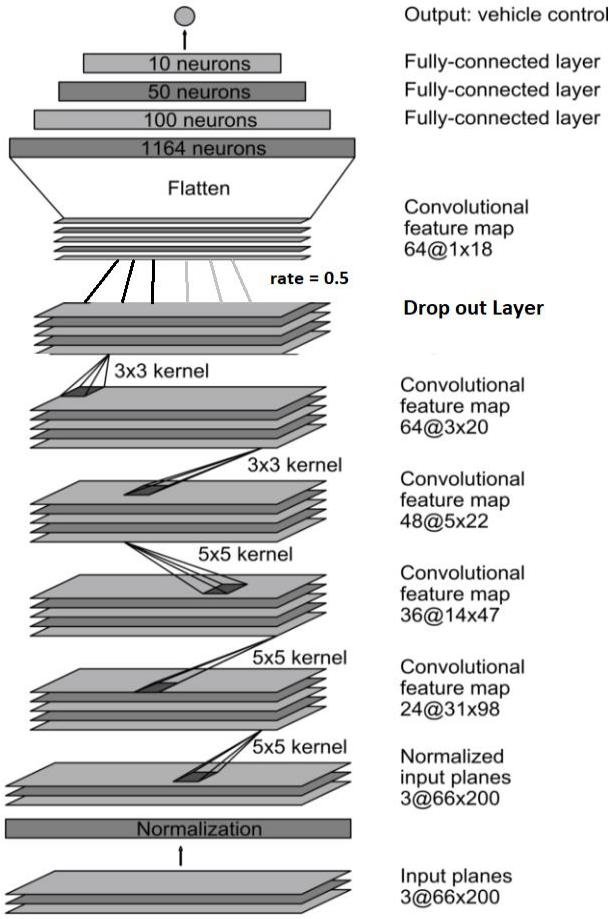

Figura 4.11.: Arquitectura de la Red Neuronal Convolucional finalmente implementada.

Para este problema se hizo uso del modelo de Red Neuronal Convolucional adaptado a un simulador de nuestras características. Esta Red Neuronal Convolucional consta de 10 capas, incluida una capa de normalización Lambda, 5 capas convolucionales, una capa de drop out con índice(0.5) y 3 capas completamente conectadas. La imagen de entrada se divide en planos YUV y se pasa a la red.

-*Entrenamiento del modelo*: Antes de entrenar el modelo, es necesario configurar el proceso de aprendizaje. Usamos la función de pérdida basada en el error cuadrático medio para medir qué tan cerca predice el modelo para el ángulo de dirección dado para cada imagen y optimizamos minimizando via descenso de gradiente.

Se entrena el modelo con los datos generados lote por lote por un generador de Python.

El generador se ejecuta en paralelo al modelo, para mayor eficiencia. Por ejemplo, esto le permite aumentar los datos en tiempo real en imágenes en la CPU en paralelo a la capacitación de su modelo en la GPU.

Finalmente se seleccionarán los distintos argumentos y parámetros de entrenamiento seleccionados son los siguientes:

-Datos destinados a Test
-Probabilidad de Dropout
-Número de iteraciones
-Muestras por época
-Tamaño de lote
-Guardar mejor iteración en cada momento
-Tasa de aprendizaje

Dichos argumentos son perceptibles a cambios dependiendo de las pruebas y validación a realizar.

4.2.2.2. Script de tratamiento de imágenes

En este apartado describiremos el script **utils.py** el cual, en términos generales, consta de diversas funciones para el tratamiento de imágenes y generación de un ángulo de giro en consecuencia.

El script está compuesto por las siguientes partes:

-*Carga de imagen*: Carga de imágenes RGB a partir de un archivo.

Figura 4.12: Ejemplo de imagen cargada.

-*Reajuste del tamaño de la imagen:* Cambiar el tamaño de la imagen a la forma de entrada utilizada por el modelo de red. Se hará uso de la función resize de OpenCV2. Ajuste a 66x200 (3 canales YUV).

-*Conversión de RGB a YUV:* El espacio de color RGB es el espacio de color más útil para las pantallas. Es un modelo de color aditivo en el que se agregan luz roja, verde y azul de varias maneras para reproducir una amplia gama de colores. YUV es un espacio de color típicamente utilizado para el procesamiento de imagen / video a color.

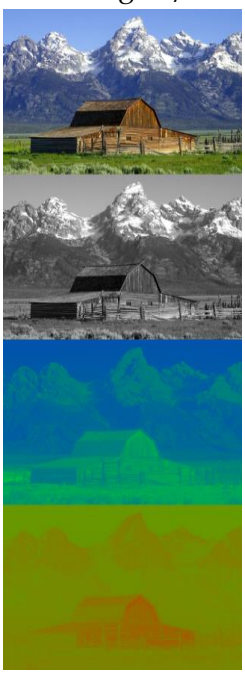

Figura 4.13: Ejemplo de Conversión de RGB a YUV

-*Elección aleatoria de imágenes y ajuste de giro:* Se escoge al azar una imagen del centro, izquierda o derecha, y se ajusta el ángulo de dirección. Para la imagen de la izquierda, el ángulo de dirección se ajusta en +0.2 mientras que para la imagen de la derecha, el ángulo de dirección se ajusta en -0,2.

-*Volteo aleatorio de la imagen:* Al azar, se voltea la imagen a la izquierda <-> derecha y ajuste el ángulo de dirección.

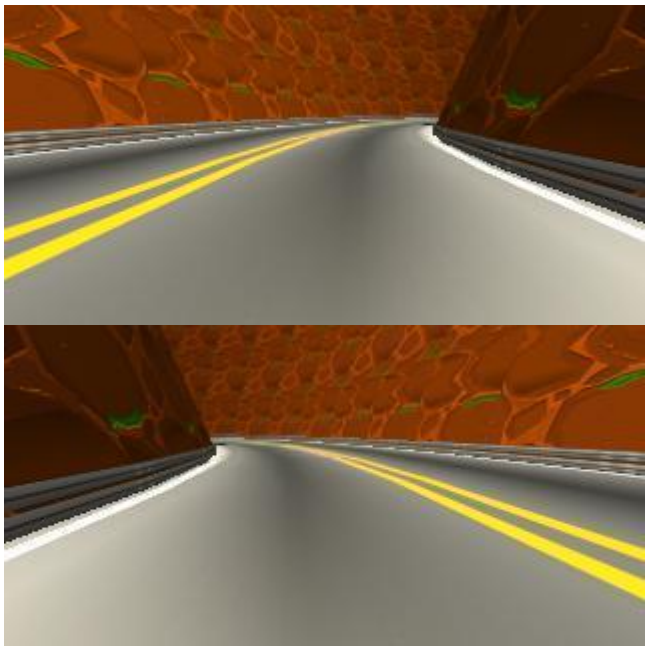

Figura 4.14: Ejemplo de imagen volteada.

-*Traducción aleatoria de la imagen:* Se traduce la imagen horizontalmente con el ajuste del ángulo de dirección (0.002 por cambio de píxel). A continuación se traduce la imagen al azar verticalmente.

-*Incremento de la imagen:* Genera una imagen aumentada y ajusta el ángulo de dirección. (El ángulo de dirección está asociado con la imagen central). Finalmente en este proceso se realizan los tres procesos anteriormente mencionados:

1)*Elección aleatoria de imágenes y ajuste de giro*
2)*Volteo aleatorio de la imagen*
3)*Traducción aleatoria de la imagen.*

Como finalidad última del script, generar la imagen de entrenamiento a la cual se le asocia la ruta del fichero donde se encuentra en local y el ángulo de giro correcto. Se realizará el *Incremento de la imagen* con todas las imágenes posibles en un mismo lote hasta ocupar todo el espacio disponible del mismo.

4.2.2.3. Script de conducción

Constituido por el script **drive.py**. A partir de aquí, se crea el servidor Python que hará conexión vía websocket con el cliente, en este caso nuestro Simulador en Unity. El servidor mandará datos de giro y aceleración en todo momento al cliente para pilotar el vehículo acorde al resultado del entrenamiento, pasándole como segundo parámetro el modelo entrenado con los pesos asignados.

Recabará los datos de cliente actuales desde el script en C# **CommandServer.cs** y los enviará vía socket al servidor en Python, nuestro script **drive.py**.

Partiendo del ángulo actual de giro, la aceleración actual (como de fuerte se está presionando el acelerador con valores de 0 a 1), la velocidad en el momento (de 0 a 30 Kmh) y la imagen de la cámara central, se pretende predecir el nuevo ángulo de giro.

De este modo, se le envía de nuevo al cliente los comandos y datos de movimiento que queremos que tenga el coche para un determinado momento, capturado por una imagen.

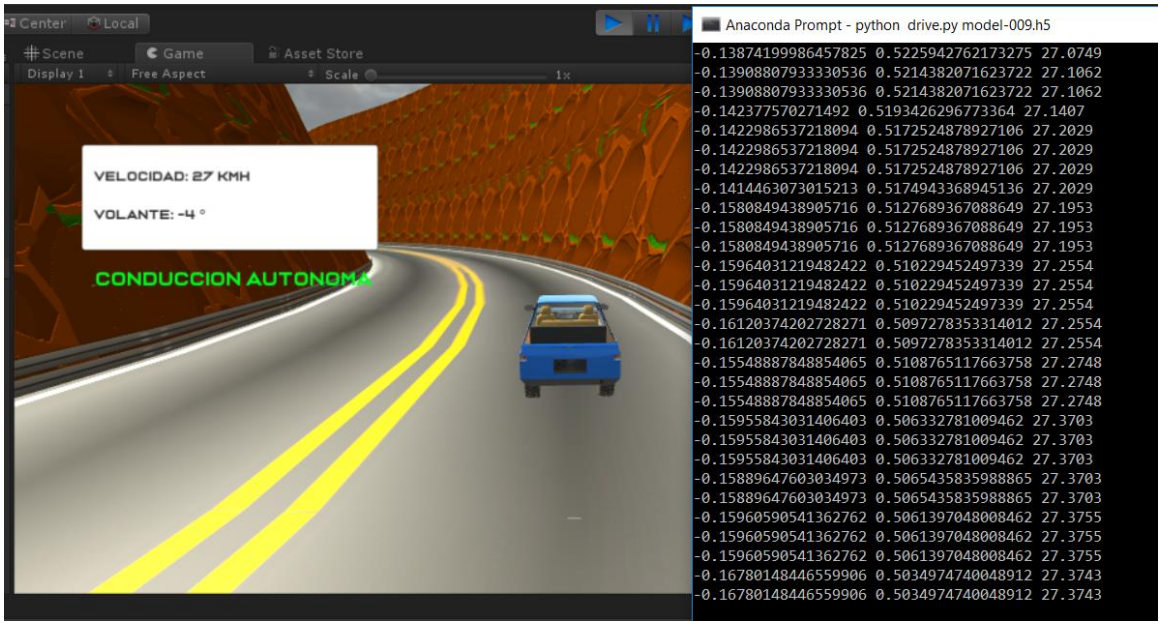

Figura 4.15: Ejemplo de conducción en Modo Autónomo. Se muestra por consola el ángulo de giro de (-1,1), la aceleración y la velocidad actual.

Capítulo 5

Pruebas y evaluación

Resumen: en este capítulo explicaremos algunas de las distintas pruebas que hemos realizado del programa y los algoritmos. Como objetivo ofrecer una serie de resultados frutos de las pruebas.

5.1 Pruebas y resultados

Finalizado el desarrollo llega el momento de someter a una serie de pruebas el simulador. Para comprobar el funcionamiento correcto del sistema hemos optado por una serie de pruebas sencillas

5.1.1 Prueba 1

Prueba	1
Descripción	Entrenamos el vehículo conduciendo de forma manual solo hasta la cuarta curva del circuito
Velocidad : 30 km/h	Nº Iteraciones: 10

Resultado	**Éxito**
Nª de muestras obtenidas	1759
Tiempo de entrenamiento de la red	640 segundos
Mejor iteración	7
Vehículo autónomo accidentado	No
Tiempo transcurrido hasta el accidente	Nunca accidentado
Velocidad del vehículo autónomo	20-25 km/h

5.1.2 Prueba 2

Prueba	2
Descripción	Entrenamos el vehículo conduciendo de forma manual hasta completar la primera curva
Velocidad : 30 km/h	Nº Iteraciones: 15

Resultado	**Negativo**,entrenamiento insuficiente
Nª de muestras obtenidas	624
Tiempo de entrenamiento de la red	994 segundos
Mejor iteración	10
Vehículo autónomo accidentado	Si
Tiempo transcurrido hasta el accidente	78 segundos
Velocidad del vehículo autónomo	25-27 km/h

5.1.3 Prueba 3

Prueba	3
Descripción	Entrenamos el vehículo conduciendo de forma manual hasta completar la segunda curva
Velocidad : 30 km/h	Nº Iteraciones: 15

Resultados	**Positivo,** aunque con el tiempo se sale de su carril
Nª de muestras obtenidas	912
Tiempo de entrenamiento de la red	945 segundos
Mejor iteración	15
Vehículo autónomo accidentado	No
Tiempo transcurrido hasta el accidente	No accidentado, sin embargo se sale de su carril habitual en la última curva del circuito
Velocidad del vehículo autónomo	22-24 km/h

Capítulo 6

Conclusiones y trabajo futuro

Resumen: en este capítulo exponemos las conclusiones extraídas durante el desarrollo del proyecto, así como alguna posible mejora que podría implementarse para ampliar la funcionalidad del sistema o mejorar su eficiencia.

6.1 Conclusión

Tras haber finalizado el desarrollo de este proyecto llega el momento de realizar una valoración de lo conseguido y lo aprendido.

El objetivo principal de este proyecto era la implementación de un simulador 3D de un vehículo autopilotado. Ciertamente podemos manifestar que esto se ha cumplido a raíz de los resultados obtenidos. No obstante este no era el único objetivo , otra serie de propósitos ligados al desarrollo del proyecto también se han cumplido , como es el caso de haber podido ampliar considerablemente nuestro saber haciendo uso de ciertas herramientas y lenguajes de programación que no habíamos utilizado antes.

Durante el desarrollo del sistema también nos hemos dado cuenta de lo difícil que resulta trabajar y adaptar código de terceros, y la cantidad de tiempo y recursos que requiere entenderlo antes de poder trabajar con él.

En resumen , pese a algunas dificultades hemos conseguido realizar un proyecto que nos ha permitido poner en práctica lo aprendido y adquirir nuevos conocimientos, y dado que esta era una de las razones principales que nos llevaron a realizar este proyecto, hemos quedado muy satisfechos de lo logrado.

6.2 Conclusion

After completing the development of this project, it is time to make an assessment of what has been achieved and what has been learned.

The main objective of this project was the implementation of a 3D simulator of an autonomous vehicle. Certainly we can affirm that this has been fulfilled as shown by the results obtained. However, this was not the only objective, other series of purposes linked to the development of the project have also been fulfilled, as is the case of having been able to greatly expand our knowledge by making use of certain tools and programming languages that had not been used before.

During the development of the system we have also realized how difficult it is to work and adapt third-party code, and the amount of time and resources required to understand it before we can work with it.

In summary, despite some difficulties we have managed to carry out a project that has allowed us to put into practice what we have learned and acquire new knowledge, and since this was one of the main reasons that led us to carry out this project, we have been very satisfied with what accomplished.

6.3 Trabajo Futuro

Una vez llegado el momento de haber finalizado el desarrollo del proyecto, siempre nos preguntamos qué más cosas podrían añadirse para mejorarlo en un futuro.

Realmente es posible realizar un sinfín de mejoras a algo encaminado a la conducción autónoma de vehículos , pero de cara a mejorar este sencillo simulador y poner en práctica nuevos conocimientos citamos un ejemplo a continuación.

6.3.1 Integración de Ray casting para detección de obstáculos

El raycasting[20] es un proceso consistente en disparar un rayo invisible desde un punto hacia alguna dirección con la idea de detectar los objetos que colisionan con dicho rayo. Ha sido usado desde hace tiempo para resolver muchos problemas en el ámbito de la geometría computacional y los gráficos por ordenador.

Para poder introducir un raycast(o varios) en el simulador , Unity nos brinda ya un componente , *Physics.Raycast,* que permite lanzar un rayo desde un punto origen en una dirección dada por un vector director , pudiendo además ajustar la distancia máxima que debe alcanzar el rayo.

Dentro de la estructura general del proyecto este raycast haría las veces de sensor de distancia y al igual que el resto de sensores, sus valores en cada muestreo durante el entrenamiento serán almacenados en el archivo CSV, y con algunas leves modificaciones del script de entrenamiento podría ser tomado como un valor más para entrenar la red neuronal.

De esta forma cuando el usuario entrene el vehículo y esquive los obstáculos manualmente cuando se encuentre a cierta distancia de ellos , la red neuronal "aprenderá " de ese comportamiento.

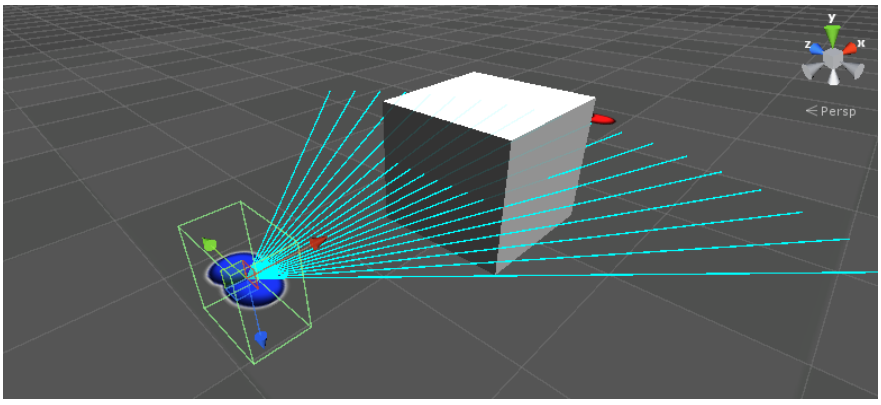

Figura 5.1: Ejemplo de conjunto de raycast en Unity 3D

6.4 Aspectos éticos futuros

Si se consigue que todos nuestros vehículos sean autónomos entonces se supondría haber conseguido un perfecto tráfico dónde los accidentes serían casi nulos, salvo imprevistos ocasionados por peatones, animales u otros elementos del entorno.

Tras algunos casos de accidentes de atropello ocasionados por vehículos autónomos, se llegó a la conclusión de que los sistemas inteligentes podían fallar ante situaciones críticas de tal magnitud. Aquellas situaciones donde hay que valorar una serie de variables como que vida humana tomar en caso de que se presenten dos posibilidades. Por ejemplo, si desviarme y perecer el conductor o seguir recto y atropellar a los peatones. En este caso, el sistema inteligente de autoconducción debería valorar cual es la mejor opción, pero aquí se plantea el dilema moral.

Después de investigar acerca de este debate ético sobre que debe decidir el coche autónomo ante estas situaciones, encontramos que la vida humana siempre va por encima de cualquier animal u objeto, seguidamente, si se elige entre vidas humanas, aquella situación que más se ciña a las normas de tráfico y miniminice los daños.

Por otra parte, la llegada de la conducción autónoma implicaría el dejar completamente a la máquina ser amos del vehículo lo cual no es del todo conveniente ya que se pierde sensación de control y seguridad, por las razones anteriormente explicadas sobre el dilema ético. La incapacidad de los ocupantes del automóvil de tomar las riendas en caso de imprevisto o de un incidente inminente les produce esta inseguridad. Por ello lo mejor sería un sistema de conducción asistida, mayormente autónomo. Quién sabe si en el futuro, se podría evitar este tipo de situaciones y el vehículo autopilotado llegase a una solución óptima al respecto.

Capítulo 7

Contribuciones individuales

7.1. Javier Antón Alonso

Como ya hemos explicado a comienzos de este documento, la primera fase del proyecto consistió en la realización de labores de investigación. Tanto de los antecedentes del proyecto y su historia, como de herramientas y tecnologías que pudiesen resultar de utilidad. En general mi rol principal fue como Analista.

De cara a facilitar la forma en la que se desenvuelve el proceso de investigación , repartimos las tareas , en mi caso propuse utilizar Unity 3D y estudiar las posibilidades que podría darnos de cara a trabajar con él, mientras mi compañero también estudiaba la parte de funcionamiento de los scripts de aprendizaje automático.

Aunque tenía algo de experiencia de desarrollo con Unity en el campo 2D , era completamente nuevo en el apartado 3D. Gracias a la gran cantidad de tutoriales y documentación que Unity pone a disposición del usuario no fue una tarea tan compleja como podría parecer a priori.

Una vez completada la fase de aprendizaje con Unity llevamos a cabo una búsqueda de posibles assets que sirvieran de utilidad para diseñar nuestro simulador. Una vez estaban ya elegidos tras analizar entre Xuebo y yo que cumplían lo que buscábamos, procedimos a realizar pruebas con los mismos. Dedicamos bastante tiempo a entender y comprender los scripts y las jerarquías de funcionamiento de cada uno de los assets, dado que nuestra implementación consistiría en adaptaciones y extensiones de los mismos.

Tras completar estas fases llegó el turno de la implementación del simulador. Diseñamos las escenas que componen el simulador y comprobamos que todos los elementos que las componían funcionaban según lo esperado, para que posteriormente mi compañero pudiera testearlas al ponerlas en funcionamiento con los de scripts en Python.

Una vez comprobamos el funcionamiento correcto del sistema realizamos una serie de pruebas para probar la eficiencia del sistema, algunas de las cuales son mostradas en esta memoria. Durante la realización de estas pruebas mi función fue la de tomar los datos a medida que mi compañero las iba ejecutando.

Finalmente me encargue junto a mi compañero de la redacción y síntesis de esta memoria, así como en la corrección de errores ortográficos y conceptuales.

7.2. Xuebo Zhu Chen

En lo que respecta a mi contribución, partiendo de la idea conjunta de realizar un simulador de conducción autónoma en 3D, realicé tareas de investigación hasta llegar a un modelo del que podíamos partir. Asumí un rol de Programador.

Se decidió utilizar Unity 3D como motor gráfico para poner en funcionamiento una serie de scripts en Python que implementaran algoritmos de deep-learning para autos de conducción autónoma. Partí de una serie de scripts, los cuales se inspiraron en una solución que proponía NVidia para este tipo de problemas.

Una vez lo anterior había quedado decidido, desde un primer momento dividimos el proceso de investigación entre Javier y yo. Por mi parte investigué todas las tecnologías y librerías que debíamos utilizar estudiando recursos que ponían a disposición distintos investigadores y estudiantes, respecto al tema. Todos ellos de licencia libre y con similitud a nuestro proyecto. En concreto, varios simuladores de conducción.

A continuación, dediqué gran parte de mi tiempo a instalar el entorno y las dependencias para poner en funcionamiento los scripts que entrenarían un modelo y conducirían nuestro vehículo. Lo especifiqué con detalle en el Ejemplo de Funcionamiento y el Manual de Usuario.

Por ello, trabajé con Unity 2017.4.0f1 ya que posteriores versiones de Unity daban problemas con algunos Assets y Plugins. Utilicé los Assets que recomendó Javier y en consecuencia configuré el vehículo de manera que fuese capaz de recabar imágenes y mandar datos recolectados en formato csv, para su posterior procesamiento, entrenamiento y tratamiento para la posterior conducción. Ajusté los mecanismos, giro, ruedas y físicas del mismo para hacerlo funcional y así identificar las variables asociadas a dichas funciones (Giro, aceleración, freno y velocidad), necesario para guardar estos datos en el CSV.

Consecuentemente pasé a la sección de scripts donde estudié e intente comprender los algoritmos y el modelo que proponía NVidia para este problema, como el funcionamiento de los scripts fue elaborado por terceros, realicé únicamente algunas adaptaciones.

Tras conseguir que el sistema funcionara de forma correcta y diese los resultados esperados decidimos someterlo a una serie de pruebas para tomar nota sobre sus eficiencia, algunas de estas pruebas son las mencionadas en la memoria. Durante la realización de estas pruebas fui el encargado de ejecutar el sistema con diferentes parámetros , mientras mi compañero tomaba notas de los resultados obtenidos.

A su vez, tomé los recursos necesarios para la conducción autónoma, como lo son los plugins de SocketIO para Unity, que habilitaba la función de control remoto vía websocket y threadDispatcher para la ejecución concurrente. Configuré los scripts necesarios para el envío de comandos de manera remota cliente-servidor. Añadimos el fichero CarRemoteControl para el modo de conducción autónoma.

Finalmente, elaboré los apartados en la memoria específicos a las tareas que llevé a cabo, y junto a Javier hicimos una revisión de la misma.

Apéndice A

Ejemplo de funcionamiento

En este apartado expondremos un ejemplo de funcionamiento de nuestra herramienta junto a la ejecución de los scripts.

Comenzaremos por ejecutar nuestro archivo .exe correspondiente al Simulador 3D. Seguido nos aparecerá una ventana de configuración de varios aspectos gráficos y comandos para el control del coche.

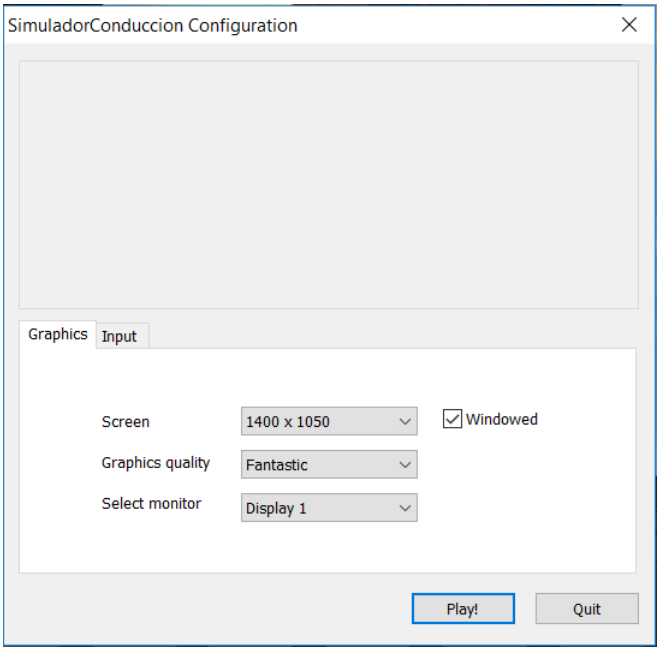

Figura A.1: Diálogo de configuración inicial de la aplicación

A continuación nos aparece la pantalla de Menú Principal donde podremos seleccionar los distintos modos de nuestro simulador. En este caso, primero seleccionaremos el Modo Entrenamiento para entrenar nuestro vehículo en el circuito, grabando el recorrido para la posterior construcción del modelo. Véase Figura 4.9.

Para ello pulsaremos en el botón Entrenar. Se abrirá un diálogo donde seleccionaremos en que carpeta queremos que guarde tanto las imágenes captadas del recorrido que hagamos como el csv con los datos referentes al mismo.

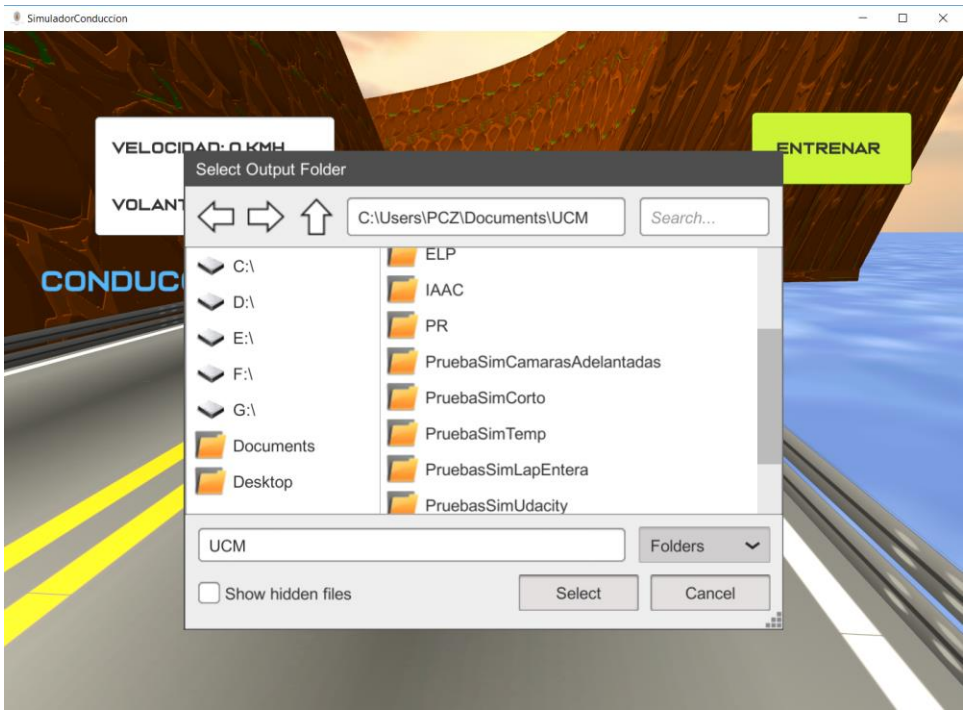

Figura A.2: Diálogo mostrado para la selección del directorio destino al guardar los datos recopilados del recorrido.

En el siguiente paso realizaremos el recorrido del circuito con normalidad, intentando siempre permanecer en nuestro carril. En nuestra prueba mantuvimos el acelerador al máximo (30 Kmh) permaneciendo siempre en el carril correspondiente.

Figura A.3: Conducción en Modo Entrenamiento y toma de curva en el circuito.

Tras finalizar el recorrido de primera vuelta completa detendremos la grabación presionando en Detener Entrenamiento. Consecuentemente se escribirán los datos correspondientes en local. Durante el proceso se mostrará el recorrido realizado a medida que se escriben los datos.

Figura A.4: Ejemplo de Escritura en Disco una vez Finalizado el entrenamiento y recorrido manual.

Como resultado obtendremos un directorio IMG con el total de imágenes del recorrido y el CSV con los datos característicos del vehículo.

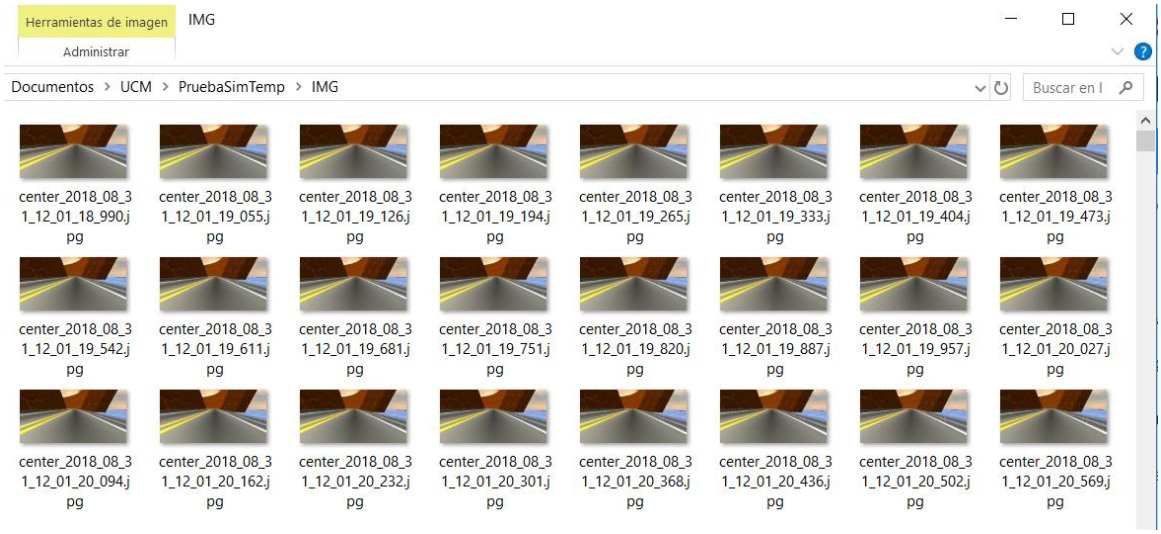

Figura A.5: Directorio con las imágenes capturadas del recorrido.

Seguidamente, ejecutaremos los scripts en Python desde el Prompt de Anaconda . Para ello primero hay que activar el entorno en el que se va a ejecutar. Si queremos utilizar TensorFlow sin GPU ejecutar `conda env create -f entorno.yml` sin embargo si queremos utilizar TensorFlow con GPU ejecutar `conda env create -f entorno-gpu.yml` (*Véase Manual de Usuario si da fallos a la hora de crear el entorno*). Si se dispone de una GPU en condiciones se es altamente recomendable utilizarla ya que agiliza el proceso de entrenamiento y procesado de imágenes. En mi caso, al disponer de una NVidia Geforce GTX 1050 Ti podemos explotar su capacidad de procesamiento para el entrenamiento de imágenes de la red neuronal. En el siguiente apartado Manual de Usuario se hará especial hincapié en este aspecto.

A continuación, nos situaremos en el directorio específico el cual contiene todos nuestros scripts y activamos el entorno de ejecución.

```
Successfully installed keras-1.2.0 moviepy-0.2.3.5 protobuf-3.6.1 pyyaml-3.13 tensorflow-gpu-1.1.0 theano-1.0.2 tqdm-4.25.0
#
# To activate this environment, use
#
#     $ conda activate entorno-coche-autonomo
#
# To deactivate an active environment, use
#
#     $ conda deactivate


(base) C:\Users\PCZ\Xuebo\TFG\Model_Utils_Drive_py>activate entorno-coche-autonomo

(entorno-coche-autonomo) C:\Users\PCZ\Xuebo\TFG\Model_Utils_Drive_py>
```

Figura A.6: Activación del entorno de ejecución en Anaconda

Una vez creado y activado el entorno podemos pasar a construir y entrenar el modelo con el siguiente comando, `python model.py`. El tiempo estimado de espera para un entreno de la red neuronal con 10 iteraciones y 20000 muestras por iteración es de unos 10-15 minutos para un total de casi 20000 imágenes.

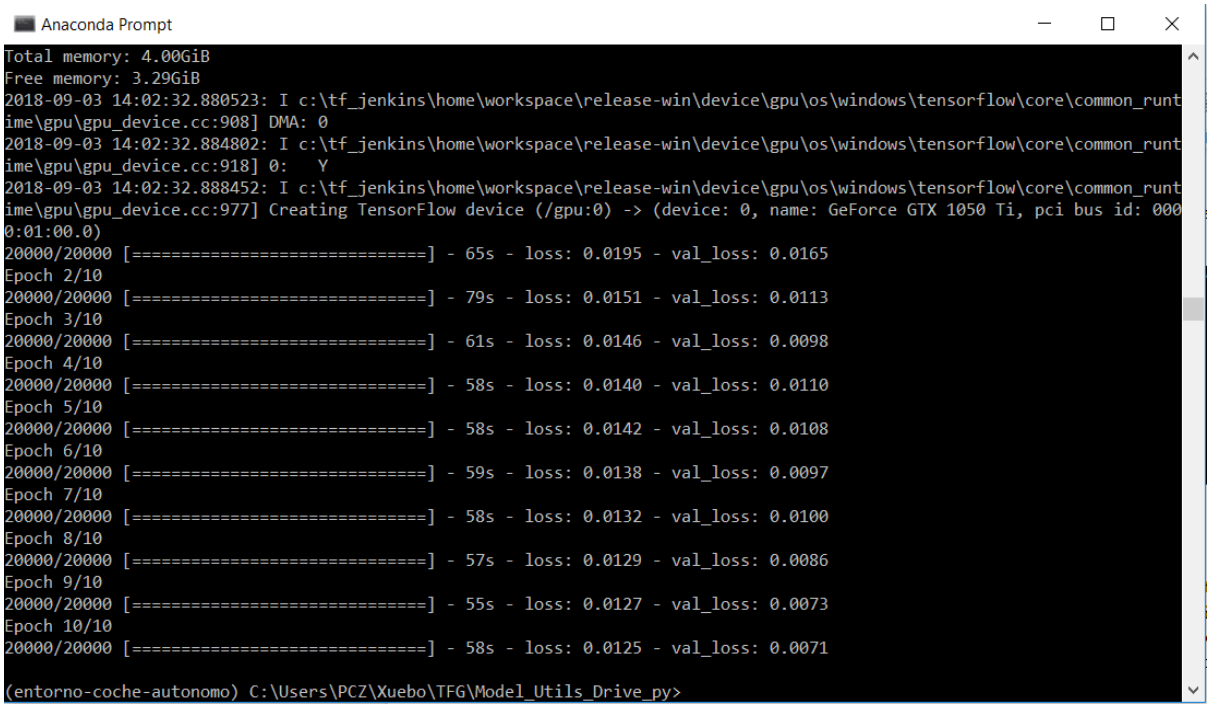

Figura A.7: Ejemplo de ejecución de model.py con 10 iteraciones

Al tener activado el parámetro save_best_only no se sobrescribirá el último mejor modelo de acuerdo con la cantidad monitoreada, por lo que si una iteración dió mejor resultado que las anteriores se guardará como un nuevo fichero 'model-{n_iteración}.h5'.

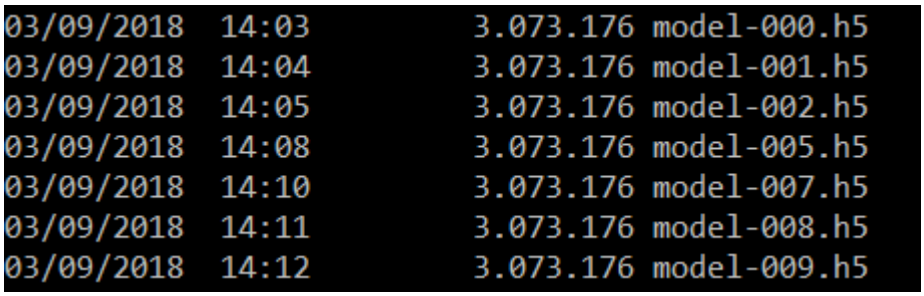

Figura A.8: Se guarda el modelo después de cada iteración (epoch) siendo el último el que mejor rendimiento ha dado.

Por último, seleccionaremos el Modo autónomo en nuestro simulador. Una vez iniciado el modo autónomo pasaremos a la consola de Anaconda y ejecutaremos el script de conducción con:

```
python drive.py model-009.h5
```

Figura A.9: Ejemplo de ejecución en Modo Autónomo ejecutando drive.py con el modelo entrenado.

Apéndice B

Manual de usuario

Enlace al repositorio: https://github.com/xuebozhu/TFG
Resumen: en este apéndice explicaremos aspectos de interés para la instalación y uso del sistema implementado.

Para empezar, es necesario descargar el .exe correspondiente a nuestro simulador o abriendo y compilando el proyecto de Unity.Una vez lo tengamos simplemente hay que ejecutarlo y seleccionar la configuración inicial. *Véase Figura A.1 del Ejemplo de Funcionamiento.*

A la hora de ejecutar los Scripts de Python correspondientes, hay que tener en cuenta que hay dos formas de entrenar nuestro modelo. La primera no haciendo uso de la GPU, sólo de la CPU y la segunda con GPU. Los tiempos entre ambos tipos de entrenamiento distan bastante ya que para el procesamiento de imágenes y el entrenamiento de la red neuronal con estas es más eficiente si hacemos uso de una GPU capaz. En el siguiente párrafo concretamos el entrenamiento y conducción con el uso de GPU, ya que con CPU no se necesitan de requisitos adicionales.

Se requiere del uso imprescindible de Anaconda3-5.0.1. En concreto esta versión ya que es compatible con la versión de Python que queremos utilizar, Python 3.5. Otra versión de Python no será compatible a la hora de crear nuestro entorno de ejecución, ya que muchas de las librerías que se indican en `entorno-gpu.yml` sólo funcionan con Python 3.5.

```
name: entorno-coche-autonomo
channels:
    - https://conda.anaconda.org/menpo
    - conda-forge
dependencies:
    - python==3.5.5
    - numpy
    - matplotlib
    - jupyter
    - opencv3
    - pillow
    - scikit-learn
    - scikit-image
    - scipy
    - h5py
    - eventlet
    - flask-socketio
    - seaborn
    - pandas
    - imageio
    - pip:
        - moviepy
        - tensorflow-gpu==1.1
        - keras==1.2

```

Figura B.1: Dependencia de librerías en el fichero entorno-gpu.yml

Dentro de Anaconda instalaremos Python 3.5 con el siguiente comando `conda install python=3.5.` Llegados a este punto, podremos crear el entorno, como se menciona anteriormente en Ejemplo de Funcionamiento, si queremos utilizar TensorFlow sin GPU, situarnos dentro del directorio donde se encuentren los scripts y ejecutar `conda env create -f entorno.yml` sin embargo si queremos utilizar TensorFlow con GPU ejecutar `conda env create -f entorno-gpu.yml.`

La versión de TensorFlow utilizada para el entrenamiento con GPU es `tensorflow-gpu==1.1,` la única compatible con Python 3.5, al igual que `keras==1.2.`

IMPORTANTE: Es necesario instalar los drivers necesarios de tu GPU (Tarjeta Gráfica) para poder realizar la construcción y entrenamiento del modelo. Características de nuestra GPU. *Véase GPUs supported.* https://en.wikipedia.org/wiki/CUDA, para las distintas versiones de CUDA para cada GPU:

Nvidia GeForce GTX 1050 Ti specs

- CUDA Cores: 768
- Base Clock: 1290MHz
- Boost Clock: 1392MHz
- Texture Units: 48
- Memory Clock: 7GHz
- Memory Bandwidth: 112GB/s
- ROPs: 32
- L2 Cache Size: 1024KB
- Transistors: 3.3 billion
- Die Size: 132mm2
- Process: Samsung 14nm FinFet
  Versión CUDA == CUDA SDK 8.0 support for compute capability 2.0 – 6.x

Seguidamente hay que descargarse la librería cuDNN la cual se ha instalado la versión cuDNN v6.0 (April 27, 2017), para CUDA 8.0. Para su instalación, copiar los siguientes ficheros dentro del directorio CUDA Toolkit.

1. Copia `<installpath>\cuda\bin\cudnn64_6.dll` en `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0\bin`.
2. Copia `<installpath>\cuda\ include\cudnn.h` en `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0\include`.
3. Copia `<installpath>\cuda\lib\x64\cudnn.lib` en `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0\lib\x64`.

Asegurarse de que los siguientes valores están asignados dentro de Variables del Sistema/Entorno:

1. **Variable Name**: CUDA_PATH
2. **Variable Value**: C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0.

Aquí resumiré las tecnologías y herramientas necesarias para la ejecución completa de nuestra herramienta.

Requisitos:

- **Simulador.exe (Aplicación)**
- Anaconda3-5.0.1.
- Python 3.5
- *Numpy*
- *Matplotlib*
- *Jupyter*
- *Opencv3*
- *Pillow*
- *Scikit-learn*
- *Scikit-image*
- *Scipy*
- *H5py*
- *Eventlet*
- *Flask-socketio*
- *Seaborn*
- *Pandas*
- *Imageio*
- *Moviepy*
- *Keras==1.2*
- *Tensorflow == 1.1*
  Con GPU además se necesita:
  - *Tensorflow-gpu==1.1*
  - CUDA 8.0
  - cuDNN v.6.0

Nota: En cursiva aquellas librerías que se instalan automáticamente al crear el entorno desde `entorno.yml` o `entorno-gpu.yml`

Bibliografía

[1] "Automovil",Wikipedia. Disponible en:

https://es.wikipedia.org/wiki/Autom%C3%B3vil

[2] Daniel Matus, "Historia de los carros autónomos contada en unos pocos hitos", Digital Trends . Disponible en:

https://es.digitaltrends.com/autos/historia-carros-autonomos/

[3] "Vehículo Autónomo", Wikipedia. Disponible en:

https://es.wikipedia.org/wiki/Veh%C3%ADculo_aut%C3%B3nomo

[4] John McCarthy,"Computer Controlled Cars". Disponible en:

http://www-formal.stanford.edu/jmc/progress/cars/cars.html

[5] Janosch Delcker, "The man who invented the self driving car" , Politico. Disponible en:

https://www.politico.eu/article/delf-driving-car-born-1986-ernst-dickmanns-mercedes/

[6] Dean Pomerleau, "Defense and civilian applications of the ALVINN robot driving system". Disponible en:

https://www.ri.cmu.edu/pub_files/pub3/pomerleau_dean_1994_1/pomerleau_dean_1994_1.pdf

[7] Python. https://www.python.org/

[8] Unity. https://unity3d.com/es/learn

[9] Manuel Zaforas,"Tensorflow y el futuro de la inteligencia artificial según Google", Paradigma. Disponible en:

https://www.paradigmadigital.com/dev/tensorflow-sera-futuro-la-inteligencia-artificial-segun-google/

[10] "Numpy and Scipy documentation".Disponible en:

https://docs.scipy.org/doc/

[11] "Why use Keras?"https://keras.io/why-use-keras/

[12] "Pandas documentation". Disponible en :

http://pandas.pydata.org/pandas-docs/stable/

[13] "H5F5 for Python" disponible en:

http://docs.h5py.org/en/stable/

[14] Patricio Loncomilla,"Deep Learning: Redes Convolucionales" disponible en:

https://ccc.inaoep.mx/~pgomez/deep/presentations/2016Loncomilla.pdf

[15] Mariusz Bojarski, Davide del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, Karol Zieba | April 25, 2016. *End to End Learning for Self-Driving Cars*.

https://arxiv.org/pdf/1604.07316.pdf

[16] Marcos Schultz,"MSVehicleSystem ". Disponible en:

https://assetstore.unity.com/packages/tools/physics/ms-vehicle-system-free-version-90214

[17] SpybloodGames,"Envionmental Race Track Pack".Disponible en:

https://assetstore.unity.com/packages/3d/environments/roadways/environmental-race-track-pack-63493

[18] Fabio Panettieri,"SocketIO for Unity".Disponible en:

https://assetstore.unity.com/packages/tools/network/socket-io-for-unity-21721

[19] Naoki Shibuya | Feb 15, 2017. *Car behavioral cloning*.

https://github.com/naokishibuya

[20] "Raycasting".Unity . Disponible en:

https://unity3d.com/es/learn/tutorials/topics/physics/raycasting

[21] Zhilu Chen ; Xinming Huang |2017. *End-to-end learning for lane keeping of self-driving cars.*

[22] GPUs supported. https://en.wikipedia.org/wiki/CUDA

[23] Install Tensorflow GPU on Windows using CUDA and cuDNN. https://www.codingforentrepreneurs.com/blog/install-tensorflow-gpu-windows-cuda-cudnn/