ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

**GRADO EN INGENIERÍA DEL SOFTWARE**

**Estudio de la conducción automática simulada aplicando técnicas inteligentes**

**Study of simulated automatic driving by applying intelligent techniques**

Realizado por

**Olivier Gabana Gómez**

Tutorizado por

**Eduardo Guzmán de los Riscos**

Departamento

**Lenguajes y ciencias de la computación**

UNIVERSIDAD DE MÁLAGA

MÁLAGA, FEBRERO DE 2025

Fecha defensa: 7 de julio de 2025

Resumen

En este proyecto se busca encontrar las limitaciones de la últimamente popularizada Inteligencia artificial, en este caso será aplicada al mundo de la conducción autónoma, y la metodología puesta a prueba será el aprendizaje por refuerzo.

Para realizar este proyecto se hará uso de una herramienta llamada Carla, la cual nos permitirá crear un mundo en el cual se encontrará el vehículo autónomo, para así poder entrenarlo y encontrar sus limitaciones.

En primera instancia se entrenará al vehículo únicamente haciendo uso de sensores básicos, como lo son el sensor de colisión, cruzado de línea y obstáculos. Por lo tanto, la idea es ver hasta donde es capaz de llegar un coche que conduce "a ciegas".

En segunda instancia esta vez se le añadirá al vehículo un sensor mucho más avanzado llamado LIDAR (Light Detection and Ranging), el cual genera una imagen de lo que se encuentra delante del vehículo lanzando varios rayos láser para así poder saber a qué distancia se encuentran los obstáculos.

**Palabras clave: Aprendizaje, Refuerzo, Inteligencia, Conducción, Automatización.**

Abstract

This project seeks to find the limitations of the recently popularized Artificial Intelligence, in this case it will be applied to the world of autonomous driving, and the methodology tested will be reinforcement learning.

To carry out this project, a tool called Carla will be used, which will allow us to create a world in which the autonomous vehicle will be located, in order to train it and find its limitations.

In the first instance, the vehicle will be trained only using basic sensors, such as the collision sensor, line crossing and obstacles. Therefore, the idea is to see how far a car that drives "blindly" can go.

In the second instance, this time a much more advanced sensor called LIDAR (Light Detection and Ranging) will be added to the vehicle, which generates an image of what is in front of the vehicle by firing several laser beams to know how far away the obstacles are.

**Keywords: Reinforcement, learning, autonomous**

Índice

Introducción 1

1.1 Motivación 2

1.2 Objetivos 2

1.3 Estructura de la memoria 3

Tecnologías a utilizar 8

Carla 9

Python 3.7 y librerías 10

Metodología de trabajo 12

Introducción a las metodologías ágiles 13

Funcionamiento y etapas de las metodologías ágiles 14

Aplicación en este proyecto 15

Desarrollo del Software 17

Instalación, configuración y funcionamiento de Carla 18

Creación, y puesta en marcha del vehículo autónomo 22

Explicación bases teóricas 23

Introducción a agentes, entorno y otros conceptos básicos 23

Exploración o explotación 26

Metodologías basadas en política y en valor (Policy-Based and Value-Based Methods) 27

Aprendizaje por refuerzo profundo (Deep reinforcement learning) 29

Explicación teórica Fase 1 30

Introducción y sensores 30

Programación y configuración Fase 1 32

Resultados obtenidos Fase 1 32

Instalación (y funcionamiento) del sensor LIDAR en el vehículo autónomo 32

Explicación teórica Fase 2 32

Programación y configuración Fase 2 33

Resultados obtenidos Fase 2 33

Comparación de resultados entre fases 34

Conclusiones y líneas futuras 37

Referencias 39

Manual de uso 41

Requerimientos: 41

1

Introducción

1.1 Motivación

La motivación de este proyecto surge de la gran revolución que estamos experimentado últimamente en cuanto a las inteligencias artificiales, como por ejemplo el mundialmente conocido ChatGpt, el cual te responde de una forma más o menos acertada a cualquier pregunta que tengas sobre cualquier tema, desde un suceso histórico hasta que te resuelva una integral y te muestre el procedimiento que ha realizado. También existen otras aplicaciones de inteligencia artificial, como en las últimas generaciones de tarjetas grafica de la empresa NVIDIA, la cual hace uso de esta para generar fotogramas extras a la hora de jugar a videojuegos para así mejorar la experiencia sin necesidad de tener un procesador gráfico tan potente.

En el mundo de la automoción existe un gran caso de éxito el cual no es ni más ni menos que el *autopilot* de Tesla, el cual es en parte una gran parte de la inspiración para la realización de este proyecto.

1.2 Objetivos

Existen 2 objetivos principales a la hora de realizar este proyecto, aprendizaje del funcionamiento de la inteligencia artificial y ver las capacidades reales de esta misma.

Ya que la inteligencia artificial se encuentra en un momento de gran crecimiento y no existen todavía una gran cantidad de profesionales especializados en esta materia, se considera un conocimiento bastante valioso en el mercado laboral, por esta razón uno de los objetivos principales es el aprendizaje del funcionamiento interno de la inteligencia artificial.

Últimamente se intenta aplicar inteligencia artificial a todos los sectores, pero, ¿es realmente tan "inteligente" como parece?, esta pregunta es considerada bastante importante para así poder saber si es simplemente una innovación que estará un tiempo de moda o si ha llegado para quedarse en el mundo de la tecnología. Por lo tanto uno de los objetivos de este proyecto es ver la capacidades reales de la inteligencia artificial y si tiene la capacidad real de suplantar tareas que actualmente son solo realizadas por personas, como por ejemplo la conducción (aunque ya existen sistemas bastante desarrollados como el antes mencionado "autopilot" de Tesla), o si en cambio es algo mucho más simple de lo que pensamos y por ejemplo aplicado al caso de la conducción solo sería capaz de conducir en trazadas muy simples sin una gran cantidad de tráfico.

1.3 Estructura de la memoria

En primer lugar, se hará una explicación sobre todos las herramientas y tecnologías utilizadas para el desarrollo de este proyecto, para así poder contextualizar correctamente toda la explicación sobre el desarrollo de software.

En este apartado sobre las herramientas utilizadas, se hará un resumen sobre la herramienta Carla, la cual nos permite simular todo el entorno en el que se desenvolverá nuestro vehículo autónomo, Carla permite crear tráfico artificial, cambiar entre ciudades, alterar el clima, y muchas otras funciones que hacen posible que se le dedique la mayoría del tiempo al desarrollo de la inteligencia artificial en sí.

También se hablará de todas las herramientas utilizas para el desarrollo del "cerebro" de nuestro vehículo autónomo, principalmente librerías de Python, sin las cuales el desarrollo del software sería mucho más complejo.

En segundo lugar, se hablará sobre la metodología de trabajo que se ha utilizado durante todo el desarrollo del proyecto, es este caso la metodología utilizada será una metodología de tipo ágil.

*Una* ***metodología ágil*** *es un enfoque flexible e iterativo para la gestión de proyectos y desarrollo de software que prioriza la colaboración, la entrega continua y la adaptación al cambio. Se basa en principios del* ***Manifiesto Ágil****, como la comunicación cercana con el cliente, la mejora continua y la entrega de valor en ciclos cortos (sprints). Ejemplos comunes incluyen* ***Scrum, Kanban y XP (Extreme Programming)***

En tercer lugar, se hablará de todo el desarrollo del software del proyecto, se dividirá en las diferentes fases en las cuales se ha desarrollado el proyecto. En cada una de estas fases se explicarán todas las decisiones tomadas desde el principio hasta el final de la fase, para así poder entender todas las decisiones de diseño y modificaciones que se han ido haciendo a lo largo del desarrollo de la fase, para así poder obtener los mejores resultados posibles.

En la primera fase, como antes se ha mencionado, solo se hará uso de sensores más "simples", como lo pueden ser el sensor de colisión, cruzado de línea, ... Y en vez de usar una red neuronal con una alta complejidad, solo se hará uso de aprendizaje por refuerzo ( no profundo), por lo que entonces se hará uso Qtables las cuales serán tratadas con más profundidad más adelante. El objetivo de esta fase es ver hasta que punto con un sistema lo más "sencillo" posible, es capaz de conducir un coche autónomo.

En la segunda fase, se añadirá a todo nuestro set de sensores un nuevo sensor, el cual es el LIDAR (Light Detection and Ranging), el cual es un sensor mucho más complejo, y es capaz de generar imágenes con puntos de colores que simulan un láser, para así poder saber la distancia a la que se encuentran los objetos del alrededor el vehículo.

Además, en esta fase no se hará uso de aprendizaje por refuerzo como en la fase anterior, sino que se hará uso de aprendizaje por refuerzo profundo, el cual es un método bastante más complejo que el anterior, gracias al cual deberíamos de obtener mucho mejores resultados que con el anterior. Sin embargo, no son todo ventajas, ya que la complejidad del desarrollo del software aumenta bastante, al igual que en la anterior fase, se tratará más en profundidad más adelante para así poder comprender las diferencias.

En último lugar se analizarán los resultados obtenidos, para así poder comparar las metodologías utilizadas, y ver si realmente merece la pena crear una red neuronal compleja. En este apartado también se tratará la pregunta planteada al principio (¿es realmente tan "inteligente" como parece?), ya que es una de las principales motivaciones de la realización de este proyecto.

2

Tecnologías a utilizar

Carla

Para la realización de este proyecto se hará uso del antes mencionado Carla, un software que nos permitirá generar un mundo con todo el tráfico simulado y añadir nuestro coche autónomo para poder entrenarlo.

Existen otros Framework y programas que también nos permitirían realizar el proyecto, como por ejemplo: LGSVL, NVIDIA DRIVE Sim, Microsoft AirSim; pero, ¿Porque hemos elegido Carla y no otro Software?, a continuación se enumerarán las mayores ventajas que tiene Carla frente a los otros Software que se encuentran en el mercado:

1. Carla es gratuito lo cual es una gran ventaja, ya que reduce el coste de desarrollo considerablemente frente a otros simuladores como lo puedo ser NVIDIA DRIVE Sim.
2. Otra gran ventaja de Carla es que nos aporta una simulación muy realista frente a otros simuladores como por ejemplo LGSVL, el cual no tiene una mala calidad, pero no es comparable a la de Carla ya que es mucho más completa incluyendo por ejemplo fenómenos meteorológicos, sombras más detalladas, ...
3. Carla soporta una mayor cantidad y calidad de kit de sensores, ya que incluye sensores avanzados como el LIDAR, Cámaras de profundidad, GPS... Por lo tanto, en este aspecto es el claro ganador frente a otros AirSim, el cual está más enfocado en drones que en vehículos, y frente a LGSVL, el cual no tiene un kit de sensores tan completo.
4. La integración nativa de TensorFlow, PyTorch y OpenAI Gym le da una gran ventaja frente de AirSim, Carla trae integrados estos elementos ya que esta pensado precisamente para el entrenamiento de coches aútonomos, al contrario de AirSim.
5. Carla es Open-Source, lo cual es una ventaja de agradecer, ya que puedes modificar la simulación en todo lo que desees, al contrario que NVIDIA DRIVE Sim, que no puedes modificar nada en absoluto de la simulación.
6. La cantidad de información que puedes encontrar en la web (desde vídeos, webs, foros, documentación en la página oficial) es abrumante en comparación con los otros Software, ya que gracias a las ventajas que han sido mencionadas anteriormente, es la mejor opción a elegir, hay una gran cantidad de personas que comentan resultadas y muestran como ellos han configurado su coche autónomo para que pueda circular solo. Cabe mencionar la gran documentación que se puede encontrar en la página oficial de carla, ya que no solo muestra las funciones que tienen las clases de los actores que hay en la simulación, sino que también te guía paso a paso para poder arrancar y configurar tu simulación sin problemas.

Por estas razones es por la que hemos elegido a Carla frente a otras alternativas que se pueden encontrar en el mercado.

Python 3.7 y librerías

El lenguaje en el que se programará nuestro vehículo autónomo será sin duda alguna Python, ya que es el lenguaje más preparado para la programación de inteligencia artificial con diferencia.

3

Metodología de trabajo

Introducción a las metodologías ágiles

Como antes se ha mencionado en el apartado de *Estructura de la memoria*, el proyecto será desarrollado en base a una metodología ágil, la cual nos permitirá cierta flexibilidad a la hora de desarrollar el proyecto, ya que esta nos permite modificar cosas que han sido realizadas al principio el proyecto, pero que a lo largo del desarrollo de fases posteriores suponen un problema y deben de ser cambiadas para así poder terminar de alcanzar todos los objetivos del proyecto.

Las metodologías ágiles no solo tienen esta ventaja de poder modificar algo que en primera instancia no hemos resuelto de la mejor manera posible con respecto a otros tipos de metodologías usadas en el pasado como por ejemplo la metodología en cascada, la cual entre otras cosas también tenía como desventaja la falta de interacción con el cliente, sino que también, tiene muchas otras ventajas como, por ejemplo:

* Es posible realizar cambios en los requisitos a lo largo del proyecto, lo cual es una gran ventaja, ya que, si a lo largo del desarrollo aparece una nueva idea de gran utilidad o un cambio de algún requisito que no se planteó correctamente con anterioridad, se podría modificar sin grandes dificultadas y sin que supusiese un gran costo para el proyecto.
* Iteraciones cortas (con respecto a metodologías no ágiles como la metodología en cascada), lo que supone una ventaja, ya que al final de cada iteración se debe revisar todo lo que se ha hecho y se pueden plantear modificaciones y posibles nuevas funcionalidades para poder alcanzar las metas deseadas.
* Continuo Feedback del desarrollo, no solo los desarrolladores se comunicarán entre ellos al final de cada sprint, para que así todo el mundo este al tanto del proyecto, sino que también se mantendrá continuamente comunicado al cliente de como está avanzando el desarrollo del proyecto, para así poder corregir cualquier posible malentendido con el cliente que se haya producido en fases tempranas y poder corregirlo cuanto antes.
* Todas las ventajas antes mencionadas, que en general nos ayudan a poder abordar los problemas de forma temprana no solo nos ayuda a ahorrar tiempo, lo que se traduce en un ahorro del coste total del proyecto, ya que se necesitaran menos horas de trabajo para poder realizarlo. Por lo tanto, otra gran ventaja de esta metodología serían los menores costes de desarrollo.

Funcionamiento y etapas de las metodologías ágiles

Para explicar como son las fases de las metodologías ágiles, vamos a hacer una breve explicación sobre el funcionamiento de una de las metodologías ágiles más aplicadas actualmente la cual es Scrum, y es en la que está basada la metodología que se va a emplear en el este proyecto. Antes de comenzar con las fases, cabe recalcar que cada miembro perteneciente al grupo desarrolla un rol en concreto (Jefe de proyecto, responsable de calidad,...) para que así cada persona se centre en un aspecto en concreto.

La primera fase de todas es la creación del product backlog, en esta fase el jefe de proyecto se encargará de obtener todos los requisitos necesarios para el desarrollo del proyecto.

Una vez obtenidos todos los requisitos del proyecto (por el momento), se organizarán todos los Sprint, estos sprint son ciclos de entre 1 y 2 semanas, en cada Sprint se insertaran las tareas que se deben de realizar en ese periodo de tiempo, habrá breve reunión diaria para que los trabajadores expongan todas las dificultades o dudas que estén encontrando, y al final de cada sprint habrá una reunión más extensa en la cual se podrán modificar requisitos si es necesarios, ya que es una metodología ágil.

En el momento que todos los sprints esten completos, significara que no quedan tareas por realizar en el product backlog, por lo tanto, el proyecto estará finalizado.

Aplicación en este proyecto

Para el desarrollo de este proyecto, al ser solo 1 desarrollador, no se podrá exprimir al máximo esta metodología, aún así, es imprescindible el uso se esta ya que aporta una gran flexibilidad a la hora del desarrollo.

En el product backlog se encontrarán todos los objetivos que se desean alcanzar, los cuales son:

* Configuración del simulador Carla
* Creación del coche autónomo
* Acoplamiento de los sensores simples y lectura de sus valores
* Creación de la primera fase IA, para conducción autónoma
* Testeo y configuración de parámetros primera fase IA
* Entrenamiento final primera fase IA
* Acoplamiento del LIDAR al coche autónomo y procesamiento de la información
* Testeo y configuración de parámetros segunda fase IA
* Entrenamiento final segunda fase IA
* Comparación de resultados

Las tareas serán repartidas en varios Sprint, y cada sprint debería de ser una duración aproximada de 1-2 semanas.

Al final de cada sprint se revisará todo el trabajo hecho y se incluirán más requisitos si se considera necesario.

4

Desarrollo del Software

Instalación, configuración y funcionamiento de Carla

Para comenzar con la explicación del desarrollo del proyecto, hay que empezar con una explicación del funcionamiento de la herramienta principal de la que se va a hacer uso, la cual es Carla.

Vamos a empezar explicando como es la instalación de este Software. Hay que dirigirse a su repositorio de GitHub (https://github.com/carla-simulator/carla/releases) y descargar la versión deseada, en nuestro caso será la versión 0.9.15.

Una vez descargada la versión deseada, hay que descomprimir el archivo que hemos descargado, el cual nos generará una carpeta con ciertos archivos y otros subdirectorios.

El principal archivo en el que estamos interesados es CarlaUE5.exe, el cual es un archivo ejecutable que nos ejecutará el servidor de la simulación y nos generará una ventana, que no es más que un espectador mediante el cual podremos movernos y ver lo que está ocurriendo en nuestro servidor a tiempo real. Cabe recalcar que este servidor y su espectador consumen una gran cantidad de recursos computacionales, en especial recursos gráficos. Los requisitos recomendados para la ejecución son los siguientes:

* Al menos 6 GB de VRam, lo recomendado es 8 GB
* Sistema linux o Windows
* Al menos 20 GB de espacio libre de almacenamiento
* Se recomiendan 32 GB de RAM
* Un procesador de al menos 4 núcleos a una velocidad de 2.5Ghz

En la carpeta en la que extrajimos la aplicación también tenemos el proyecto de UnrealEngine del cual fue renderizado nuestra simulación, por lo tanto si se desea modificar cualquier cosa de la simulación o incluso crear nuevos mapas o vehículos se puede realizar sin ningún tipo de problemas, lo cual es una gran ventaja como se mencionó en capítulos anteriores.

Además de la todo lo que es la simulación en sí, también tenemos un conjunto de scripts de ejemplo en Python (en el directorio /PythonAPI/examples) que nos permiten realizar varias pruebas para ver como se comporta la simulación, uno de los más útiles es manual\_control.py el cual nos generará una ventana con un vehículo el cual podremos manejar con nuestros periféricos y que además no da toda la información que se puede obtener a partir de los sensores básicos que se le pueden añadir al vehículo.

A continuación, se enumerará las funcionalidades principales que presenta Carla, para que así se puedan comprender las limitaciones y capacidades de este (cabe recalcar que todas estas configuraciones se hacen desde scripts Python):

* Para poder conectarnos al servidor y ejecutar todas las acciones necesarias debemos de ejecutar las siguientes líneas de código:

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

las cuales nos permiten crear el cliente para posteriormente poder realizar cambios en nuestra simulación.

* Para poder obtener el mundo que se está simulando actualmente tenemos el comando: Forma, Rectángulo

  El contenido generado por IA puede ser incorrecto.
* Como antes se ha mencionado, no existe un solo mundo que podemos simular, sino que existe una amplia variedad para las diferentes necesidades que pueden tener los usuarios, incluso si se desea una mayor selección de mapas se puede descargar gratuitamente un paquete de extensión en el cual podemos encontrar más mundos para la simulación. Si se desea echar un vistazo a todos los mapas disponibles, se encuentra una previsualización en la siguiente URL, la cual pertenece a la documentación de carla https://carla.readthedocs.io/en/0.9.15/tuto\_first\_steps/ , en el apartado llamado Choose your map. Para poder cambiar el mundo de la simulación es necesario ejecutar el siguiente comando: **Forma, Rectángulo

  El contenido generado por IA puede ser incorrecto.**

todos los mapas disponibles se deben encontrar en el directorio "HDMaps".

* Si se desea también se puede modificar la meteorología de la simulación, para obtener la meteorología actual es mediante el siguiente comando:

**Forma, Rectángulo

El contenido generado por IA puede ser incorrecto.**

si se desea establecer cierta meteorología se puede realizar de la siguiente manera.

**Forma, Rectángulo

El contenido generado por IA puede ser incorrecto.**

* Existe la posibilidad de grabar lo que ocurre en la simulación desde un punto de vista en concreto, esto se puede gracias al comando :

**Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.**

* En el caso de tener recursos insuficientes para la ejecución fluida del servidor, existen varias opciones para poder ejecutar la simulación, la primera de ellas es ejecutar la simulación con una menor calidad, para ello a la hora de ejecutar el servidor es necesario hacerlo de la siguiente manera desde el terminal,

./CarlaUE4.sh -quality-level=Low.

También es posible no renderizar el servidor, con lo cual se ahorrarían una gran cantidad de recursos gráficos, para realizar esto es necesario ejecutar el script llamado config.py con la opción --no-rendering, este script lo podemos encontrar en la carpeta examples antes mencionada.

* **Imagen que contiene Rectángulo

  El contenido generado por IA puede ser incorrecto.**Para obtener todos los posibles actores, ya sean vehículos o sensores, necesitamos obtener la librería con todas las blueprints, esto se hace con el siguiente comando
* Si se necesita obtener todos los puntos del mundo en los que se spawnearán los actores, si se ejecuta el siguiente comando se obtendrá una lista con todos los posibles puntos

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

El respawn y configuración, ya sean NPCs o nuestro vehículo autónomo, se puede considerar como configuración del servidor, pero al ser algo más específico e importante, se explicará más detenidamente en los próximos apartados.

Creación, y puesta en marcha del vehículo autónomo

Una vez tenemos arrancado y configurado nuestro servidor, es el momento de spawnear todos los actores de nuestra simulación.

Primero, antes de spawnear nuestro vehículo autónomo, debemos de spawnear todos los NPCs (non-playable character, es decir, vehículos que forman parte de la simulación y que se mueven solos de forma autónoma y aleatoria) para así poder dotar de realismo a nuestra simulación.

Existen varias formas de spawnear estos vehículos y peatones. La primera de ellas es haciendo uso del generate\_traffic.py (que podemos encontrar en la carpeta examples) acompañado de -n para especificar el número de vehículos, y -w para especificar el número de peatones, en caso de que no se especifique alguno de ellos se spawnearán 30 vehículos y 10 peatones. La segunda opción que tenemos es añadirlo en nuestro propio script, para spawnear los vehículos primero hay que saber en que punto queremos que aparezca el actor, y después para que aparezca en el mapa hay que ejecutar el siguiente comando,

**Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.**

(en este caso será un vehículo aleatorio en un punto aleatorio), y para que el vehículo circule de forma aleatoria alrededor del mapa es necesario el siguiente comando.

**Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.**

En el caso de los peatones es necesario spawnear el propio peatón, el cual se hace igual que el vehículo anterior, pero además hay que spawnear un controlador que lo maneje, esto se hace de la siguiente manera.

**Texto

El contenido generado por IA puede ser incorrecto.**

Una vez hemos inicializado todo el tráfico de la ciudad, es hora de crear nuestro coche autónomo y añadirle todos los sensores para poder obtener toda la información del entorno que posteriormente utilizaremos para el entramiento de nuestra inteligencia artificial.

Explicación bases teóricas

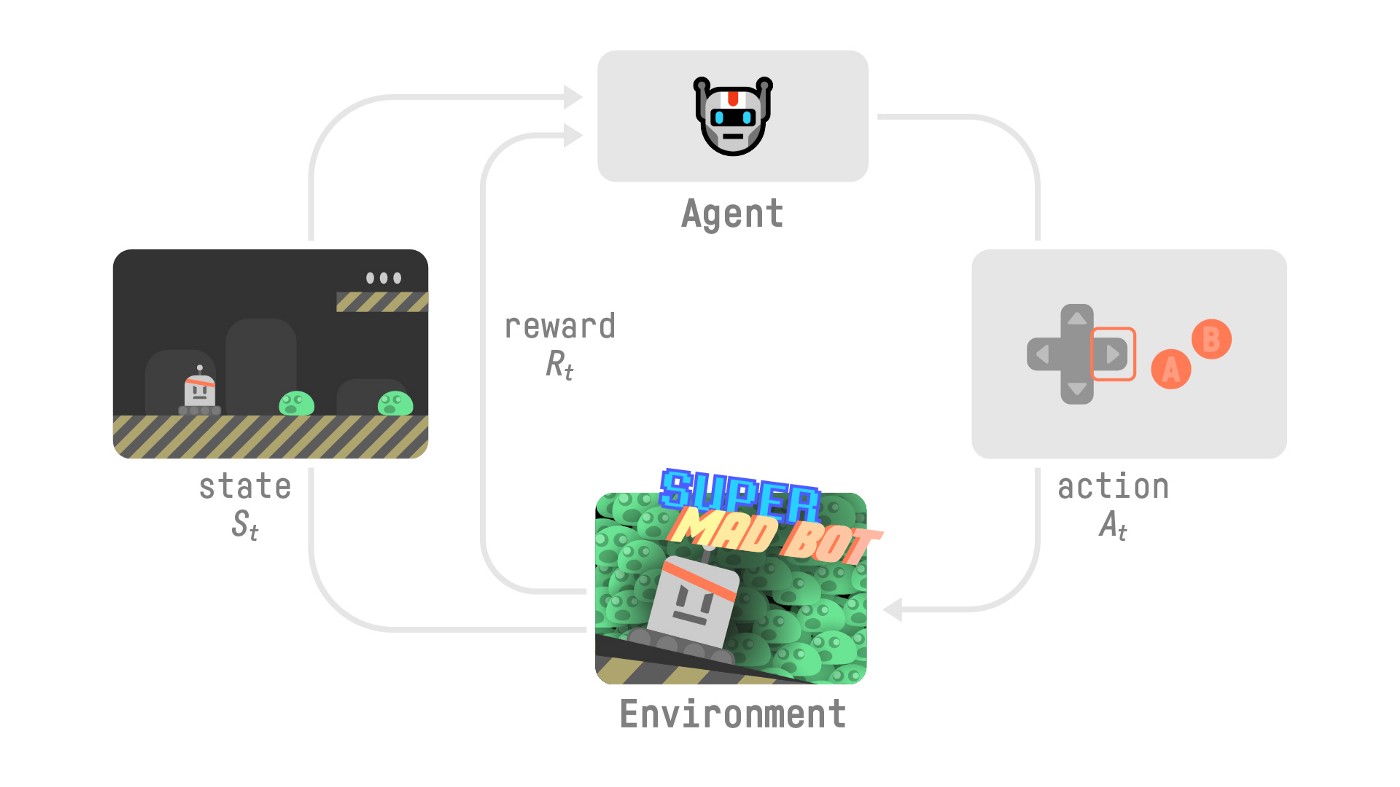
Introducción a agentes, entorno y otros conceptos básicos

Antes de comenzar con la explicación de las bases de cada una de las fases que se van a realizar, se va a realizar una breve explicación de como funciona el aprendizaje por refuerzo y de los diferentes tipos que hay, todo esto se ha con el fin de facilitar la comprensión de las siguientes fases, ya que sin unas bases correctas no se entendería correctamente porque se han tomado ciertas decisiones de diseño, por lo tanto en este apartado vamos a responder a las cuestiones más básicas que se pueden plantear sobre el aprendizaje por refuerzo y el aprendizaje por refuerzo profundo.

Primero de todo vamos a comenzar explicando que es el aprendizaje por refuerzo. El aprendizaje por refuerzo se basa en que tenemos un agente (en este caso nuestro vehículo autónomo) que interactuará con nuestro entorno ejecutando ciertas acciones y recibiendo una recompensa según lo buenas que sean estas.

Un ejemplo muy básico para facilitar la comprensión es el juego del tres en raya.

El agente sería "la máquina", y el entorno sería nuestro tablero, entonces nuestro agente iría tomando decisiones y nosotros en base a lo buenas que sean le iríamos otorgando una recompensa. Por lo tanto, nuestro agente iría aprendiendo como un ser humano, es decir, mediante la experiencia.



En la imagen anterior se puede ver como sería el proceso mediante el cual el agente va aprendiendo. Primero el agente tomaría una acción, por ejemplo, en nuestro caso del tres en raya, el agente pondría una X en una esquina. Al introducirle esa acción al entorno este actualizaría su estado, es decir, pasa de estar vacío a tener una X en una esquina, y además a parte de actualizar su estado y pasarle la información al agente, este también le envía una recompensa en base a lo buena que haya sido la acción que ha tomado. La recompensa otorgada dependerá de como haya definido la política para otorgar esta misma el programador. El objetivo de nuestro agente siempre será el de maximizar nuestra recompensa.

---------Imagen del 3 en raya representado un estado---------

Según la cantidad de información que podemos sacar del entorno, tendremos los antes mencionados estados u observaciones, a la hora de trabajar con ellos se comportan de la misma forma, pero en esencia son algo diferente y por esto es por lo que es importante tenerlo en cuenta a la hora realizar un proyecto. Lo llamamos estados cuando podemos obtener del entorno toda la información que este contiene, como por ejemplo en caso antes mencionado del tres en raya. Hay casos en el que no podemos obtener toda la información del entorno, entonces sacamos solo una parte de él, en este caso en vez de estado se llamará observación, como por ejemplo en el caso de este proyecto.

Algo parecido ocurre en el caso de las acciones que podemos tomar (también conocido como espacio de acciones), hay casos en el que solo se pueden tomar un número determinado de acciones, en este caso se dice que el espacio de acciones es discreto, como por ejemplo ocurre en el caso de nuestro tres en raya. Sin embargo hay casos en los que hay una cantidad infinita de acciones que tomar, como por ejemplo el caso de un dron, ya que puedes acelerar los motores al 1.034...%, 1.34...%, 67,4....%, ósea que hay una cantidad infinita de valores que pueden tomar los motores, por lo tanto habría una cantidad infinita de acciones. Esto es algo que hay que tener en cuenta desde el principio el planteamiento del proyecto.

A la hora de determinar la recompensa es importante tener en cuenta en que orden se han recibido, es decir las recompensas de las primeras acciones que se han tomado no deberían de tener el mismo peso que las que se toman al final, ya que es más probable recibir recompensas al principio cuando se han tomado pocas decisiones, que cuando ya se han tomado varias y se ha podido acumular una mala decisión. Por esta razón a la hora de asignar recompensas se tiene en cuenta un parámetro llamado tasa de descuento (discount rate), esta tasa debe de encontrarse siempre entre 0 y 1.

Contra más cerca de 1 se encuentre esta tasa, más en cuenta tomará decisiones tomadas al principio, esto por ejemplo sería muy interesantes en juegos como el ajedrez, en el cual un mal movimiento al principio puede comprometer toda tu partida, sin embargo, en otros casos en los que las recompensas que estamos recibiendo al final son muy importantes y no dependemos tanto de haber cometido un error al principio, utilizamos una tasa de descuento más cerca del 0.

A la hora de entrenar a nuestro agente tenemos 2 formar de hacerlo a rasgos generales, una es mediante episodios y otra es de forma continua. Si lo hacemos mediante episodios, tendremos un estado inicial, al cual después de realizar una serie de acciones, siempre acabaremos volviendo para empezar de nuevo, esto es útil en casos como por ejemplo el tres en raya antes mencionado, ya que siempre va a llegar un punto en el que se va a acabar la partida, entonces deberemos de volver al estado inicial para poder empezar otra partida y volver a entrenar a nuestro agente. Sin embargo, hay ciertos casos en los que queremos que el sistema este continuamente entrenándose, como por ejemplo un agente que se dedique a la compraventa de acciones en la bolsa, ya que esta estará continuamente comprando y vendiendo a tiempo real y estará aprendiendo siempre de las decisiones que toma.

Exploración o explotación

Antes de profundizar más en las metodologías del aprendizaje por refuerzo hay que explicar 2 conceptos bastante importantes, estos son la exploración y la explotación, y están directamente relacionados con la política que vamos a utilizar.

La explotación busca siempre obtener la mayor recompensa, es decir, con la información que tiene sobre el entorno, siempre toma las decisiones que actualmente según lo que sabe nuestro agente, le va a dar la mayor recompensa posible.

La exploración toma las decisiones de forma aleatoria, y ve como esto afecta a la recompensa que recibe nuestro agente.

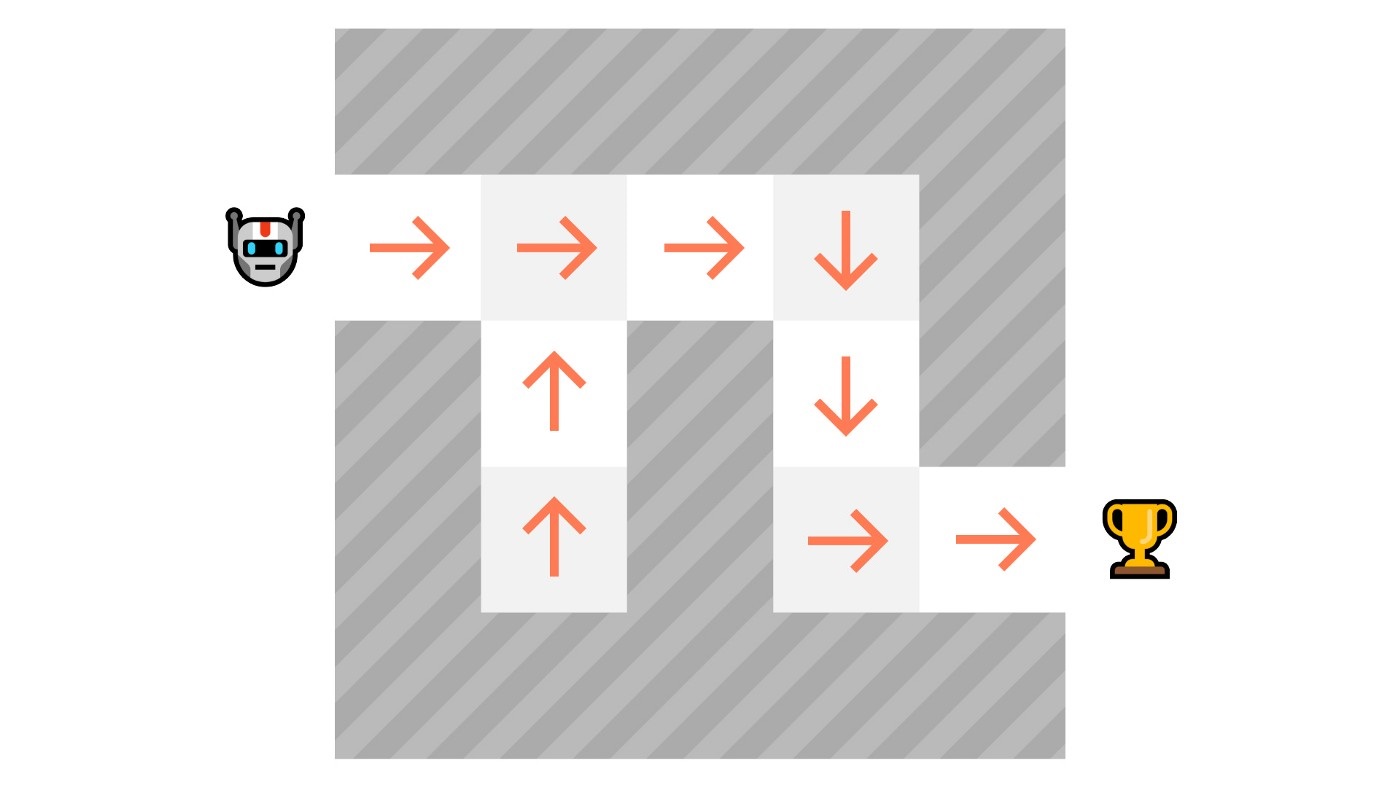
A la hora de configurar a nuestro agente, al principio este debería de usar una gran cantidad de exploración, ya que no sabemos como podemos obtener las mayores recompensas, y a lo largo del entrenamiento, ir cambiando de una estrategia que este cada vez menos basada en la exploración y más en la explotación. Así al principio exploraremos todas las opciones posibles y más o menos nuestro agente podrá ver que decisiones aproximadamente debe tomar para obtener las mejores recompensas, y al final del proceso necesitamos una mayor cantidad de explotación para así poder perfeccionar a nuestro agente y obtener el resultado final.

Aplicando este proceso, evitamos que el agente encuentre un camino que da una recompensa mediocre y que nunca salga de este, sino que al principio exploramos todas las posibilidades para que el agente no se quede estancado en un camino que le da una compensa mediocre en comparación con la que podría llegar a obtener.

Metodologías basadas en política y en valor (Policy-Based and Value-Based Methods)

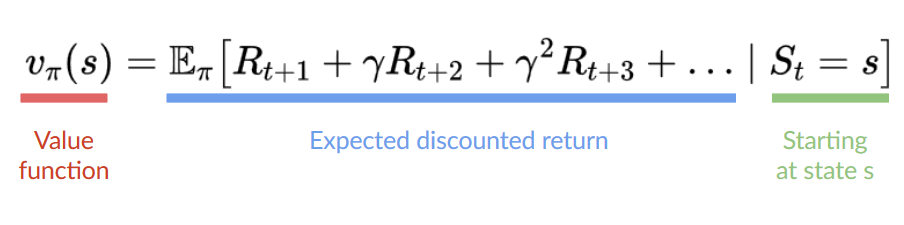
Existen 2 tipos de metodologías a la hora de entrenar a nuestro agente. La primera esta basada en editar la política de nuestro agente, la política es lo que define el comportamiento, es como el cerebro del agente, por lo tanto, es lo que decide que acciones va a tomar. En el caso de las basadas en valor, buscamos asignarle unos valores a las acciones que puede tomar el agente según en el estado en el que estén.

En la metodología basa en política, enseñamos directamente cual es la mejor acción para tomar en un estado en concreto. Es decir, modificamos la política que sigue el agente directamente.



Este tipo de política se puede aplicar de 2 formar, como se acaba de mencionar, es decir, asignando únicamente la mejor acción a cada estado (forma determinista), o también se puede hacer asignando una probabilidad a cada una de las acciones (forma estocástica).

En la metodología basada en valor no modificamos nuestra política para ir modificando el comportamiento de nuestro agente, sino que nuestro agente va aprendiendo una función valor, la cual asigna un valor a cada una de las acciones según en el estado en el que estemos. En este caso la política la definimos nosotros, ya que simplemente sería elegir la acción que nos otorgase una mayor recompensa. Para esta metodología es para la que usamos la antes mencionada tasa de descuento (discount rate).



Aprendizaje por refuerzo profundo (Deep reinforcement learning)

Hasta ahora todo lo que hemos explicado es en base al reinforcement learning. Pero existe otra variante aún más potente de la haremos uso en este proyecto, y esta es el aprendizaje por refuerzo profundo.

En el aprendizaje por refuerzo normal (no profundo), por ejemplo, en el caso de métodos basados en valor, creábamos una Q-table, es decir, una tabla en la que íbamos guardando la información que vamos obteniendo mientras entrenamos el agente, en este caso, guardamos un valor (q-value) para cada pareja de estado-acción posibles.

En el caso de el aprendizaje por refuerzo profundo no hacemos uso de la recién mencionada Q-table, sino que, para cada estado, hay una red neuronal que nos estima un valor para cada una de las acciones que podemos tomar.

Diagrama

El contenido generado por IA puede ser incorrecto.

Explicación teórica Fase 1

Introducción y sensores

Antes de comenzar con la explicación teórica de esta fase, cabe recalcar cual es la intención de esta.

En esta fase lo que buscamos es encontrar donde están las limitaciones de un sistema lo más simple posible. Por lo tanto, para este apartado, como se mencionó al principio, solo se van a hacer uso de sensores simples, los cuales serán ahora mencionados más adelante. Para crear un sistema "simple", no solo vamos a hacer uso de sensores básicos, sino que también lo va a ser nuestra "IA", para así posteriormente realizar un sistema más avanzado y poder comparar los resultados.

Los sensores utilizados en esta fase van a ser los siguientes (En el siguiente apartado llamado "Programación y configuración", se mostrará como son creados y configurados estos sensores):

* **Sensor de colisión**: este sensor envía un evento cada vez que el vehículo al que esté acoplado tiene una colisión. Este evento contiene información sobre que actores han colisionado, en qué lugar, en qué momento...
* **Sensor de invasión de línea**: cada vez que el vehículo al que esté acoplado pisa una línea de cualquier tipo, este envía un evento. Este evento contiene información sobre que qué tipo de línea es, de que color y otros atributos de menor importancia.
* **Sensor de obstáculos**: el sensor detecta otros vehículos u obstáculos a una determinada distancia hacia delante y a un radio de un valor que debe de ser especificado. Cuando detecta un obstáculo, este manda un evento que contiene a los actores, distancia...
* **Sensor RGB:** una vez que este está acoplado al vehículo, manda un evento que contiene una imagen a color de lo que se encuentra en frente del vehículo. Se puede modificar la posición de este para cambiar el punto de vista. Se puede crear una cámara a tiempo real con ayuda de este sensor.

Metodología basada en valor y decisiones de diseño

Una vez ya sabemos que herramientas vamos a utilizar de la simulación, es hora de comenzar con la explicación de nuestra IA en sí.

Para comenzar en esta primera fase no se va a hacer uso del antes mencionado aprendizaje por refuerzo profundo (deep reinforcement learning), ya que esto añadiría bastante complejidad a la solución y no es lo que se busca en este apartado, sin embargo, en el siguiente apartado si se busca un mayor nivel de profundidad y complejidad por lo que sí que se ve a hacer uso de él.

Por lo tanto, en esta fase vamos a hacer uso de aprendizaje por refuerzo normal, o sea, que no va a haber una red neuronal que nos va a dar unos valores para cada acción según el estado en el que estemos, sino que vamos a hacer uso de una metodología basada en valor, en concreto vamos a hacer uso de Q-tables.

¿Porque una metodología basada en valor?, pues porque creemos que para un sistema simple, es la mejor opción dado a las siguientes razones:

* Podemos definir claramente un espacio de acciones simplificado, y estas acciones serían: acelerar, frenar, girar a la izquierda y girar a la derecha.
* La metodología basada en valor es generalmente más simple, por lo que consume una menor cantidad de recursos computacionales en comparación con la metodología basada en política.
* Gracias a que una vez que entrenemos el modelo, este va a seguir una estrategia codiciosa, este siempre va a tomar las mismas decisiones en el mismo estado, por ejemplo, si pisa una línea interior debería de girar a la izquierda, o si detecta un obstáculo frenar. Por lo tanto, la forma de actuar sería mas determinista que en una metodología basada en política ya que normalmente esta se asigna una probabilidad a cada una de las posibles acciones en cada estado.

Entrando más en profundidad en las metodologías basadas en valor, hay 2 tipos a grandes rasgos.

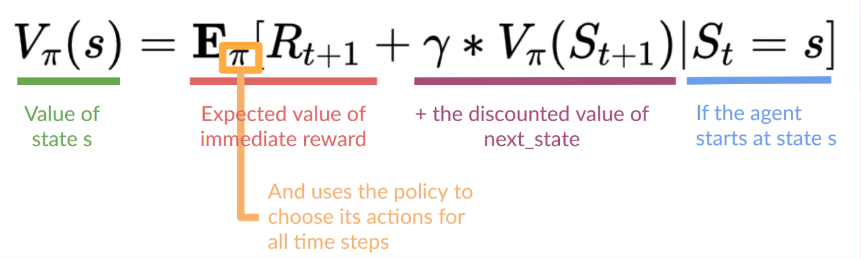
En el primer tipo únicamente le asignamos un valor a los estados, por ejemplo, si para nuestro vehículo autónomo tenemos como estados: todo bien, línea interior detectada, obstáculo detectado...; cada uno de esos estados tienen un valor que representa la recompensa que deberíamos de obtener con ese estado en caso de que todas las decisiones restantes se basasen en la política ambiciosa (más adelante se explicarán las políticas utilizadas para esta metodología). Por lo tanto, en total se almacenarán un valor por cada estado que haya (cantidad de valores = Número de estados).

En la segunda metodología no solo tenemos que un valor para cada estado, sino que tenemos un valor para cada pareja estado-acción. Por ejemplo, si tenemos las acciones acelerar y frenar, habrá un valor para acelerar-todo bien, otro para acelerar, línea interior detectada... Por lo tanto, habría que almacenar los valores en una matriz en la cual una dimensión fuesen las acciones y otras los estados (cantidad de valores = Número de estados \* número de acciones).

En el caso de este proyecto, se ha decidido optar por la segunda opción, y así asignar un valor para cada pareja, ya que, aunque aumente la complejidad del sistema, le da una gran ventaja a la hora de la obtención de resultados, ya que un coche autónomo no es en sí un sistema tan simple, y por lo tanto tener un solo valor que definiese un estado no sería suficiente para conseguir unos resultados decentes.

Herramientas y metodologías para el entrenamiento

Como se mencionó en la explicación de la metodología basada en valor, para poder darle un valor en nuestra Q-table a una pareja de estado-acción, necesitamos saber la recompensa que nos otorgarían los estados futuros (véase la imagen ------), lo que supone un reto computacionalmente hablando. Por lo que necesitamos a la ecuación de Bellman.

La ecuación de Bellman se basa en una ecuación recursiva que en vez de considerar el valor de un estado empezando desde el primer estado, asigna el valor a un estado de la siguiente manera.

Por lo tanto, ahora cuando vayamos a calcular el valor de un estado no hace falta calcular todos los siguientes, sino que utiliza el valor del siguiente estado como una estimación, ya que, el siguiente estado va a hacer lo mismo, lo que en esencia hace que sea lo mismo.

Una consecuencia a tener en cuenta cuando utilizamos esta metodología es que el valor de un estado se va a actualizar cuando sepamos cual es el estado siguiente, o sea que cuando obtenemos una observación del entorno y obtenemos un estado y una recompensa, lo que estamos actualizando es el estado anterior.

Continuando con el tema de las recompensas, hay 2 formas para asignarlas, la primera de ellas es siguiendo el aprendizaje de Montecarlo, y la otra es siguiendo el aprendizaje de diferencia temporal.

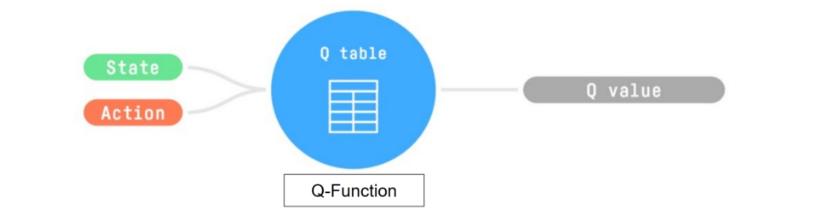
En el caso de utilizar el aprendizaje de Montecarlo, actualizaremos todos los valores de nuestra Q-table al final de cada episodio, por lo que a lo largo de los pasos iremos guardando los estados, acciones, siguientes estados y las recompensas que iremos recibiendo durante todo el episodio para poder tener todos los datos para poder actualizarlo al final. Es un método bastante útil en caso de tener episodios cortos en los que es mejor esperar que tome todas las decisiones antes de asignarle una calificación.

Sin embargo, si hacemos uso del aprendizaje por diferencia temporal, actualizaremos los valores de nuestra Q-table cada vez que se tome una acción, por lo que solo es necesario guardar el estado anterior para así poder actualizarlo. Esta metodología de aprendizaje es más útil en caso de que los episodios no estén bien definidos, como es nuestro caso, ya que no sabemos durante cuanto tiempo el vehículo va a conducir correctamente sin chocar con algo.

Por lo tanto, para el desarrollo de nuestro coche autónomo haremos uso del aprendizaje por diferencia temporal, ya que, aunque es un poco más complejo de comprender que el de Montecarlo, nos va a ayudar a la obtención de mejores resultados.

Profundización en Q-learning

Antes de pasar con el explicación práctica y mostrar como se ha realizado y configurado el código, vamos a entrar un poco más en profundidad en el campo del Q-learning, que como antes se ha mencionado, es el tipo de metodología basada en valor que vamos a utilizar en este proyecto.

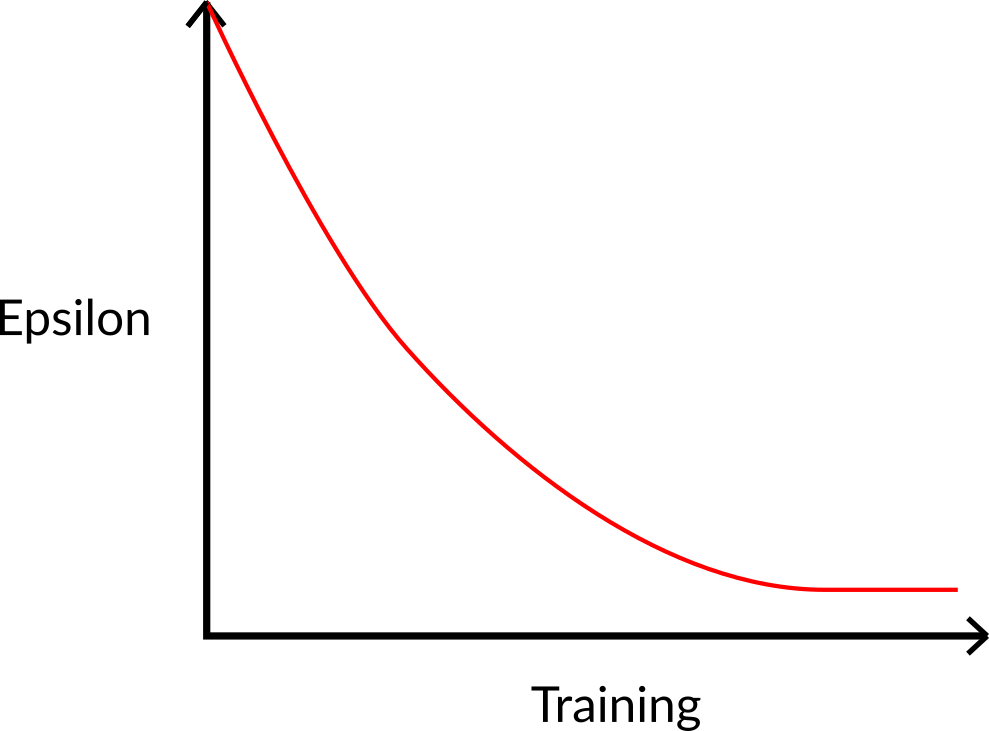


En la imagen ---------- se puede ver claramente lo que se ha explicado en los anteriores capítulos, es decir, vamos a hacer uso de una pareja Estado-Acción, para guardar el valor de esta en nuestra Q-table.

Q-learning es un método que hace uso de uso de una política diferente para tomar decisiones que para asignar la recompensas a sus parejas de Estado-Acción, a esto se le llama fuera de política (Off-policy). Otra característica de este método es que hace uso de aprendizaje por diferencia temporal, como antes se ha mencionado.

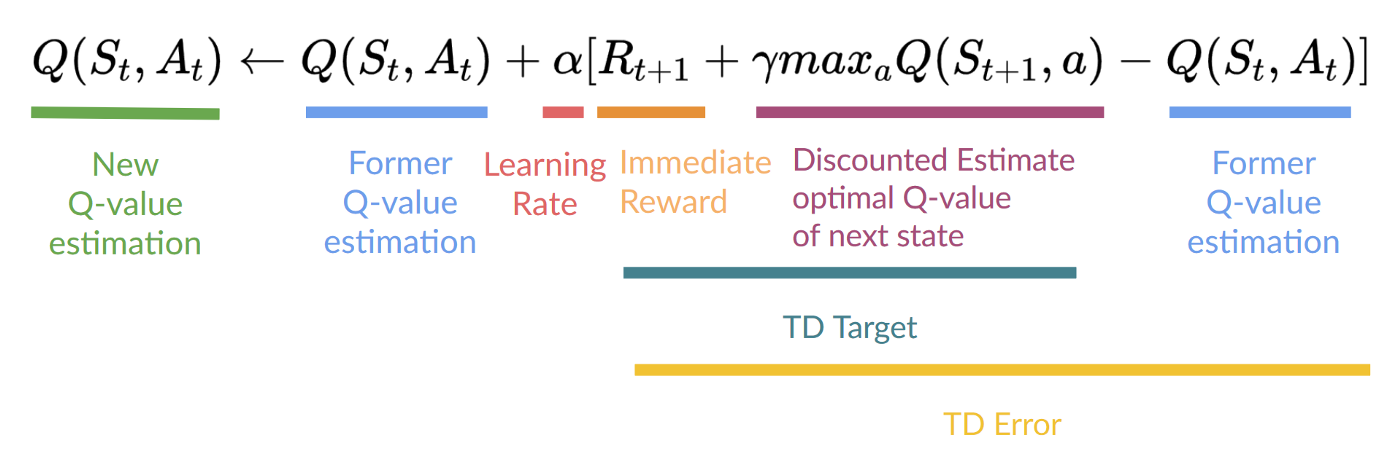
A la hora de evaluar a nuestro vehículo o de asignar el valor a nuestra pareja Estado-Acción, haremos uso de la antes mencionada en la introducción política codiciosa (Greedy), la cual es bastante simple, ya que, simplemente elige el valor que le otorga el mejor resultado posible, de ahí viene su nombre. Por lo tanto, a la hora de evaluar a nuestro vehículo, este siempre va a tomar la mejor acción para cada estado según los resultados obtenidos en el entrenamiento.

Para tomar decisiones se hace uso de una política épsilon-codiciosa (epsilon-greedy), la cual se basa en la política codiciosa. Esta política depende de un valor extra llamado épsilon, este valor oscila normalmente entre 1 y 0.05, si este valor es 0.92 por ejemplo, significa que hay un 92% de posibilidades de que se tome una decisión aleatoria, y en consecuencia un 8% de que se tome una acción haciendo uso de la estrategia codiciosa.

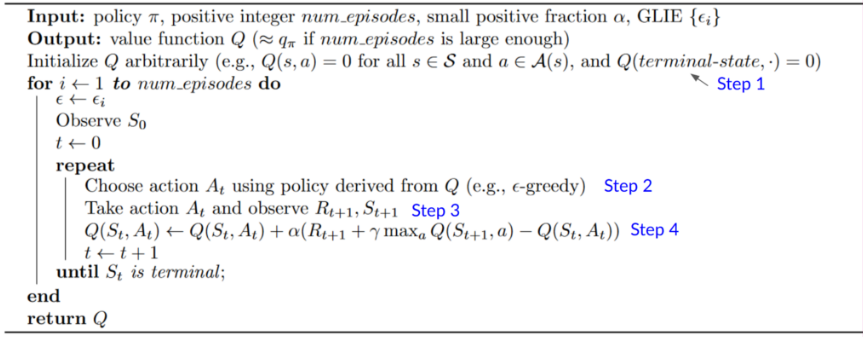
Esto es muy útil ya que nos permite controlar la explotación/exploración de nuestro vehículo, ya que al principio de entrenamiento buscamos que el vehículo pruebe todas posibilidades que hay, para que así no pierda la posibilidad de obtener una mejor recompensa de una forma diferente, pero sin embargo, según va avanzando el entrenamiento, buscamos descartar acciones que no tienen sentido y así poder centrarnos en las mejores opciones. Entonces según va avanzando el entrenamiento deberemos ir decrementando el valor de épsilon, como se puede ver en la imagen ----.

Un dato interesante a tener en cuenta que por mucho que estemos llegando al final del entrenamiento, nunca buscamos que épsilon sea 0, ya que queremos que exista una mínima probabilidad de exploración, en nuestro caso, un 5% (épsilon = 0.05).

Para asignar el valor a nuestra pareja de Estado-Acción, como acabamos de contar se hace uso de la política codiciosa, pero para su mejor comprensión, se puede ver en la siguiente imagen el funcionamiento completo de como se asigna el valor.



Formalmente hablando, el procedimiento que vamos a seguir para el entrenamiento de nuestro vehículo es el que se puede ver en la siguiente imagen -----.



* En el primer paso, inicializamos nuestra Q-table con todos los valores a 0.
* En el segundo paso, tomamos una acción según la política codiciosa, como antes de ha explicado.
* En el tercer paso, obtenemos la observación de nuestro entorno, la cual nos dice en que estado esta, y la recompensa que hemos obtenido.
* En el cuarto paso actualizaremos nuestro valor en la Q-table según la fórmula que hemos explicado.

Programación y configuración Fase 1

Inicialización de variables y creación del vehículo autónomo

En este apartado se mostrará el código utilizado para el entrenamiento de nuestro vehículo, se incluirán tanto la parte que respecta a el entrenamiento de nuestro vehículo autónomo, o sea lo que es la configuración de nuestra IA en sí, y también se incluirá todo lo que tiene que ver con la simulación, ya sea la conexión con el servidor, el spawneo de tráfico o la creación de nuestro vehículo autónomo.

Para comenzar vamos a empezar con la conexión a nuestro servidor y la inicialización de todas las variables de las cuales vamos a hacer uso.

Texto

El contenido generado por IA puede ser incorrecto.

Como se puede ver en la línea 10 de la imagen anterior, también inicializamos la clase CarlaEnv, la cual es un entorno personalizado que hereda de gymnasium (una biblioteca de Python muy que contiene entornos para ser entrenados) y mediante el cual vamos a entrenar a nuestro vehículo. Los entornos de Gymnasium, a partir de tener varias variables como por ejemplo observation\_space, también tienen todas las funciones necesarias para poder entrenar a nuestro vehículo, como por ejemplo reset(), la cual nos permite volver al estado inicial al principio de cada episodio, o la función step(), para que el vehículo tome una acción. En la línea 11, se puede ver como se llama a la función spawnearVehiculoAutonomo(), la cual no solo crea nuestro vehículo, sino que también acopla todos los sensores que se van a utilizar.

Texto

El contenido generado por IA puede ser incorrecto.

En la imagen ----, se puede ver la función completa para spawnear un vehículo. Esta comienza haciendo que el vehículo aparezca en una posición concreta, en caso de que no sea posible, se irán probando el resto de las posiciones posibles hasta dar con una que en la que sea posible. Después, seguiremos el mismo proceso para cada uno de los sensores, primero hay que buscar los planos del sensor en la biblioteca, para así poder crearlo, entonces, configuraremos todos los parámetros que nos deje el sensor. Por último, añadiremos los sensores y el vehículo a una lista para posteriormente poder eliminarlo al acabar la simulación sin problemas.

Una vez estés todos los sensores acoplados al vehículo, les diremos que comiencen a mandar información, en nuestro caso, esta información será tratada en la clase del entorno por temas de estructuración del código.

Las configuraciones de los sensores serán tratadas en el próximo capítulo, ya que han sido modificadas a lo largo de los diferentes entrenamientos. También se puede observar que el sensor de colisión no manda información, ya que, este generaba problemas (Carla es una herramienta de pruebas, y esta contiene varios errores o bugs, en especial en versiones antiguas, y estos deben de ser solucionados para el correcto entrenamiento de nuestro vehículo. Más adelante se mostrará como han sido solucionado este error) y este es configurado desde el entorno.

Especificación de parámetros para el entrenamiento y funcionamiento del Main

Después de establecer conexión e iniciar nuestro entorno, asignamos todas las variables necesarias para el entrenamiento de nuestro vehículo (aunque alguna de estas variables ya han sido explicadas anteriormente, otras todavía no se han tratado, pero se explicarán en el siguiente apartado).

Texto

El contenido generado por IA puede ser incorrecto.

Una vez que ya tenemos todos los valores para nuestro entrenamiento cargados, es el momento de spawnear el tráfico, para así poder dotar de realismo a nuestra simulación. Para poder hacer esto, llamaremos a una función que importamos de la API de Python que viene con nuestra descarga de Carla, esta API contiene un archivo ejecutable Python el cual nos generará tráfico según nosotros deseemos, por lo tanto, llamaremos a este script de Python desde nuestro main. Cabe recalcar que este script esta modificado para poder ser llamado como una función importada cualquiera de una biblioteca. Si se desea ver el funcionamiento del código, se puede ver en el archivo spawn\_npc.py. Desde nuestro main se llamara de la siguiente manera.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

El siguiente paso sería saber si queremos entrenar a nuestro vehículo, o si queremos evaluarlo, como se puede ver en la siguiente imagen.

Texto

El contenido generado por IA puede ser incorrecto.

En caso de introducir una e o E en el teclado y pulsar la tecla entrar, evaluaremos a nuestro agente, en la carpeta /TablasFase1, se encuentran varias tablas con diferentes resultados, estos resultados son los que se obtuvieron en diferentes entrenamientos con diferentes configuraciones, estos serán posteriormente explicados, pero como se puede observar en la línea 8, hay que seleccionar que Q-table queremos utilizar para evaluar a nuestro agente (las tablas estás almacenadas en un archivo tipo JSON). La función que utilizamos para cargar la Q-table (línea 8) es la siguiente.

Texto

El contenido generado por IA puede ser incorrecto.

Funcionamiento de las funciones de entrenamiento y evaluación

Ahora vamos a tratar uno de los puntos más importantes del código, y son las funciones que utilizamos para evaluar y entrenar a nuestro vehículo. Estas funciones se han separado del código principal (cliente.py) y se han puesto en un archivo aparte (FuncionesEntrenamientoFase1.py) para poder facilitar la facilidad.

Vamos a comenzar con el más simple de los 2, y este es la función que utilizamos para evaluar el agente.

Texto

El contenido generado por IA puede ser incorrecto.

Comenzamos inicializando nuestro array episode\_rewards, en el cual vamos a guardar los resultados de todos los episodios.

Después se ejecutará un bucle n veces, siendo n el número de episodios de evaluación que hemos asignado antes al principio, en nuestro caso serían 100. Al comienzo de cada iteración se reiniciará el entorno, y se inicializarán las variables necesarias. Después se ejecutará un bucle anidado, el cual iterará max\_steps veces, y en cada una de sus iteraciones, se empezará eligiendo acción según nuestra política codiciosa explicada anteriormente (siguiente imagen ----------). Después se le mandaría la acción a nuestro vehículo, y este devolvería varios atributos: información, el nuevo estado al que ha transitado, la recompensa obtenida, y si debe de acabar el episodio.

En caso de haber recibido la señal terminated, debemos de acabar el episodio, en caso contrario, guardamos el nuevo estado en la variable state. Después, al final de cada episodio, añadimos la recompensa obtenida a nuestro array episode\_rewards.

Al final del proceso calculamos la recompensa media obtenida y la desviación estándar, y devolvemos esta como resultado de la función.

Texto

El contenido generado por IA puede ser incorrecto.

Ahora es el momento de la función más importante de nuestro programa, y esta es la función train\_agent, la que utilizaremos para entrenar a nuestro vehículo.

Pantalla de computadora con letras

El contenido generado por IA puede ser incorrecto.

Toda la función se basa en un bucle que se va a repetir n\_traning\_episodes veces, en nuestro caso esa variable tenía el valor 3005. Lo primero que hace la función es reducir el valor de épsilon, para así tener cada vez menos exploración y más explotación como se explicó previamente. Después mostramos en el terminal cierta información para ir teniendo un feedback de como está avanzando el entrenamiento, ya que es un proceso bastante largo.

Antes de empezar a tomar acciones, hay que reiniciar nuestro entorno e inicializar las variables de las que haremos uso en las siguientes líneas. Cada 200 episodios, guardamos la Q-table, para así después ver como ha ido avanzando el entrenamiento.

Una vez reiniciado el entorno, ejecutaremos un bucle max\_steps veces, en nuestro caso max\_steps tiene el valor de 200. Lo primero que hacemos dentro de este bucle es tomar una acción siguiendo la política épsilon-codiciosa (se puede ver como se ha hecho el código en la imagen ---------). Una vez tomada esta acción, el entorno nos devuelve: información, el nuevo estado al que ha transitado, la recompensa y si el episodio debería de terminar. Una vez tenemos la información, actualizamos el valor de la pareja Estado-Acción, siguiendo la fórmula que se ha explicado anteriormente (se puede ver en la imagen ------).

Por último, terminamos el episodio en caso de que en entorno nos lo haya enviado en la variable "terminated", y en caso contrario, asignamos a la variable state el nuevo estado al que ha transitado el entorno.

Configuración del entorno para el entrenamiento (CarlaEnv)

Antes de acabar con esta sección, falta mostrar toda la funcionalidad de la clase CarlaEnv, la cual es un entorno que hereda de Gymnasium, pero está modificado para poder simular las acciones de nuestro vehículo autónomo.

Texto

El contenido generado por IA puede ser incorrecto.

A la hora de definir la clase, lo hacemos como se puede observar en la línea 3 de la imagen --------, para que esta herede de Gymansium.

Después podemos ver el constructor de la clase, el cual inicializa todos los parámetros a la hora de crear la clase. Un punto a tener en cuenta es que en la variable action\_space, vamos a guardar el numero de acciones posibles que puede tomar nuestro vehículo, y en la variable observation\_space, guardaremos la cantidad de estados posibles de nuestro entorno.

Para poder realizar otras funciones de la clase, tenemos una función para volver a poner al coche en la posición inicial y otra para asignar el vehículo autónomo.

Texto

El contenido generado por IA puede ser incorrecto.

En cuanto a la función que se encarga de mover el vehículo a la posición inicial, hay un par de puntos a tener en cuenta. El primero de ellos es que el vehículo aparece en un lugar diferente aleatoriamente cada vez que se ejecuta la función. El segundo, es que antes de mover el coche, se le desactivan las físicas, para que así, antes de moverlo a la nueva posición el vehículo este completamente quieto. Por último, pero no menos importante, si el vehículo tiene un sensor de colisión acoplado, este se destruye y se añade otro nuevo una vez se ha movido el vehículo a la nueva posición. Esto se hace porque cuando el vehículo cambia de posición, este cae del cielo, y entonces el sensor de colisión detecta una colisión, lo cual es un bug del simulador Carla, y para evitar que este ocurra, hay que realizar este proceso (los time.sleep también son para la prevención de bugs).

Como antes se ha mencionado, todas las señales de los sensores se manejan en el entorno, y esto se hace con las siguientes funciones.

Texto

El contenido generado por IA puede ser incorrecto.

Como se puede ver en imagen anterior, hay una función para cada uno de los sensores, vamos a realizar una breve explicación para que se entienda el funcionamiento y propósito:

* Sensor de cámara: dado que al intentar generar un vídeo con las imágenes genera cierta inestabilidad, se opta por mover el espectador a la posición del coche, por lo que el espectador va a seguir al vehículo desde arriba para que se pueda ver el funcionamiento de este.
* Sensor de obstáculos: si no hay un 1 en la cache, añade un 1 cuando ha detectado un obstáculo. El cache se lee y borra entre acción y acción. Esto se hace con el fin de saber en que estado se encuentra el vehículo y de poder asignar recompensas al vehículo.
* Sensor de colisiones: añadimos un 0 a la cache, y paramos las lecturas del sensor de colisión para evitar el error del sensor de colisión que se ha mencionado hace un momento.
* Sensor de invasión de línea: nuestro objetivo es poder discernir entre líneas discontinuas, líneas continuas exteriores, y líneas continuas interiores. Por lo que, como todas las líneas amarillas o tipos de doble continua son líneas interiores, añadimos un 3 a la cache. En caso de que sea una línea continua simple, es una línea continua exterior, por lo que añadiremos un 2 a la cache. Por último, si la línea es discontinua, añadiremos un 4 a la cache.

Estos valores que hemos añadidos a la cache, son evaluados en la función de recompensa y en la función de obtener una observación, con el fin de poder asignar una recompensa y de saber en que estado se encuentra el vehículo.

Texto

El contenido generado por IA puede ser incorrecto.

En la imagen anterior se puede ver que recompensas se le dan al vehículo según lo que hayan detectado los sensores y la velocidad y distancia que se ha desplazado. La razón de esos valores se explicará en el próximo capítulo con detalle, dado que son valores que han ido cambiando según las diferentes etapas del entrenamiento.

Al final de la función se guarda el valor de la posición actual en la variable ultimaPosicion, para así poder ver que distancia se ha desplazado la próxima vez que se llame a la función.

Texto

El contenido generado por IA puede ser incorrecto.

En la función de la imagen anterior se pueden ver los 20 posibles estados que se declararon al principio en el constructor de la clase.

Al principio de la clase se calcula a la velocidad que se está moviendo el vehículo en metros por segundo, para poder dividir cada uno de los estados en 4 estados los cuales se diferencian en la velocidad a la que se está moviendo el vehículo. Por ejemplo, si el sensor de obstáculos hubiese detectado uno, este hubiese añadido un 1 al cache, y dependiendo a la velocidad a la que estuviese moviéndose el vehículo, este estaría en el estado 0, 1, 2 o 3.

La primera vez que se implemento este método, únicamente tenía 4 posibles estados, ya que no se hacía diferencia según a la velocidad a la que se estuviese moviendo el vehículo. Pero según fue avanzando el desarrollo del proyecto, se llego a la conclusión de que por mucho que se intentase mantener el sistema lo más simple posible, era imprescindible tener en cuenta a la velocidad a la que se estaba moviendo el vehículo, ya esta es de vital importancia a la hora de tomar una decisión u otra. No es lo mismo detectar un obstáculo cuando estás parado que cuando vas a 50Km/h.

Resultados obtenidos Fase 1

Instalación (y funcionamiento) del sensor LIDAR en el vehículo autónomo

Explicación teórica Fase 2

Programación y configuración Fase 2

Resultados obtenidos Fase 2

5

Comparación de resultados entre fases

5

Conclusiones y líneas futuras

Referencias

https://agilemanifesto.org/ -- definición de metodologías agiles

Apéndice A

Manual de uso

Requerimientos: