Apprentissage par renforcement de la conduite d'un véhicule sur AirSim

Culture Sciences de l'Ingénieur

Édité le 19/07/2022

école normale supérieure paris—saclay

Ludovic DE MATTEÏS - Saša RADOSAVLJEVIC

Cette ressource est issue d'une co-publication avec le numéro 109 de La Revue 3EI de juillet 2022 et fait partie du « Dossier Intelligence Artificielle » [5] sur Culture Sciences de l'Ingénieur. *Ludovic De Matteïs et Saša Rodasavljevic sont élèves du département Nikola Tesla de l'ENS Paris Saclay.*

Cette ressource propose d'une part une application pédagogique concrète et attrayante de l'apprentissage par renforcement : l'apprentissage réaliste de la conduite autonome sur le simulateur AirSim. D'autre part, il présente l'intégration d'un environnement de simulation aux bibliothèques Python Gym et Stable Baselines, facilitant l'utilisation des algorithmes d'apprentissage par renforcement (voir ressource « *Introduction à l'apprentissage profond* » [6]).

1 – Introduction

L'émergence des transports autonomes obligent le développement de nouvelles méthodes de conduite. Plusieurs méthodes d'intelligence artificielle peuvent être utilisées afin de créer des systèmes de conduite autonome performants. Dans cette ressource, nous nous intéresserons particulièrement à l'utilisation de l'apprentissage par renforcement pour répondre à notre problème.

Cette étude étant particulièrement adaptée à une séance de travaux pratiques, la ressource s'attache à détailler les différents éléments de l'apprentissage par renforcement et leur implémentation, en utilisant des outils logiciels adaptés. Les activités peuvent être effectuées à différents niveaux d'études en s'attachant plus ou moins à la construction de chacun des éléments (agent, environnement, récompenses, observations…). Les résultats des phases d'entrainements et la progression de l'agent présentent clairement l'apport de l'apprentissage par renforcement pour la résolution de problème complexes.

Cette ressource propose une méthode d'intégration d'un simulateur dans la démarche d'apprentissage par renforcement. Cette méthode peut être réutilisée et adaptée à d'autres problèmes d'apprentissage pour lesquels un simulateur est accessible. Cette ressource ne détaillera pas les concepts de l'apprentissage par renforcement mais s'orientera seulement vers leur mise en œuvre dans le cadre du problème de conduite autonome. Les aspects théoriques de l'apprentissage par renforcement peuvent être retrouvés dans d'autres ressources du « *Dossier Intelligence Artificielle* » *[5].*

2 – Déroulement de la séance de travaux pratiques

2.1 - Objectif de la séance

Cette séance d'activité pratique a pour objectif de présenter les concepts d'apprentissage par renforcement par un exemple complexe concret.

La problématique centrale de cette séance est celle de la conduite autonome. La complexité de la tâche est telle qu'il est difficile de la résoudre de manière performante avec des outils classiques de planification de trajectoire et d'évitement d'obstacles. Nous allons alors nous intéresser à la faisabilité de l'utilisation d'outils d'intelligence artificielle pour résoudre, même partiellement, ce problème. La disponibilité d'un simulateur rend le problème très adapté à une résolution par apprentissage par renforcement.

Cette séance commencera par présenter les éléments fondamentaux de l'apprentissage par renforcement, à savoir la définition de l'espace des observations, de l'espace des actions et de l'agent. Par la suite, l'implémentation de l'algorithme d'apprentissage sera présentée en utilisant les bibliothèques Python Gym et Stable Baselines. La séance est constituée de quelques questions ramenant aux concepts de bases d'apprentissage et d'étapes permettant de mettre en œuvre l'exemple étudié.

2.2 - Support de l'activité

Cette séance d'activité pratique se basera sur l'utilisation du simulateur AirSim[1]. Il s'agit d'un simulateur de drone et de voiture intégré au moteur physique Unreal Engine 4[2] (il existe également une intégration au moteur physique Unity, encore en développement). L'utilisation d'un simulateur pour cette activité permet de travailler sur un problème concret même si l'accès ou l'utilisation du système réel est impossible. Dans le cadre de la conduite autonome l'utilisation d'un système réel n'est pas envisageable en raison des nombreuses collisions qui auront lieu lors de la phase d'apprentissage. Enfin, en utilisant les fonctionnalités d'AirSim et d'Unreal Engine, il est possible d'accélérer le temps en simulation et donc d'apprendre plus rapidement. Ce dernier point est fondamental pour l'application des algorithmes d'apprentissage en séances de travaux pratiques, les phases d'apprentissage étant généralement longues.

Dans le simulateur, la voiture par défaut est utilisée, à laquelle est ajouté un capteur LIDAR de paramètres réglables. Le paramétrage de la simulation est fait par un fichier « *settings.json* » se trouvant dans les documents utilisateur (l'installation du simulateur AirSim est décrite plus en détail dans la partie suivante).

2.3 - Outils utilisés

Question 1 :

Parmi les algorithmes de Reinforcement Learning que propose Stable Baselines, lequel semble le plus adapté selon vous (justifiez) ?

Réponse 1 :

L'algorithme Advantage Actor-Critic (A2C) peut être utilisé pour avec un espace des actions importants, cependant son utilisation favorise le multi-agent pour apprendre avec plusieurs environnements en parallèle, ce qui peut être contraignant niveau ressources. L'environnement simulé que nous utilisons demandant déjà beaucoup de ressources, cet algorithme ne semble pas adapté. L'agorithme Deep Q-Learning (DQN) demande un espace d'action discret mais propose d'utiliser un espace des observations important permettant la prise en compte des nombreuses mesures que peut procurer le Lidar. Cette méthode est bien adaptée à notre problème et nous utiliserons donc le DQN dans le projet présenté. L'algorithme Hindsight Experience Replay (HER)

[1] https://microsoft.github.io/AirSim/
[2] https://www.unrealengine.com/en-US

simplifie le système de récompense et utilise d'autres politiques d'apprentissage. Il peut être intéressant d'approfondir cette méthode afin d'observer ses performances.

Description

Pour réaliser l'apprentissage par renforcement, il faut définir les éléments principaux. Ceux-ci sont représentés figure 1.

- L'agent, prenant des décisions en fonction des observations reçues, est modélisé et géré par la bibliothèque Python Stable Baselines. C'est le conducteur autonome de la voiture. Il s'agit dans notre cas d'un réseau de neurones de type Multi Layer Perceptron (MLP) comportant 2 couches cachées de 64 neurones. La bibliothèque utilisée permet également de superviser la phase d'apprentissage en assurant le lien entre l'agent et l'interpréteur.
- L'interpréteur (ou environnement d'apprentissage), permettant l'envoi d'observations de l'environnement simulé à l'agent, la définition des récompenses et leur envoi à l'agent et l'envoi d'actions de l'agent vers l'environnement simulé. Dans l'activité présentée ici, l'interpréteur sera réalisé à l'aide de la bibliothèque Python Gym.
- L'environnement simulé, comme discuté précédemment, permet de faire évoluer l'agent dans le circuit simulé en fonction des actions choisies et de lui retourner les informations des capteurs. Il comprend la voiture, ses capteurs et actionneurs, la piste, et leurs interactions. L'environnement est créé par le moteur physique Unreal Engine 4 et le simulateur AirSim y intègre la voiture.

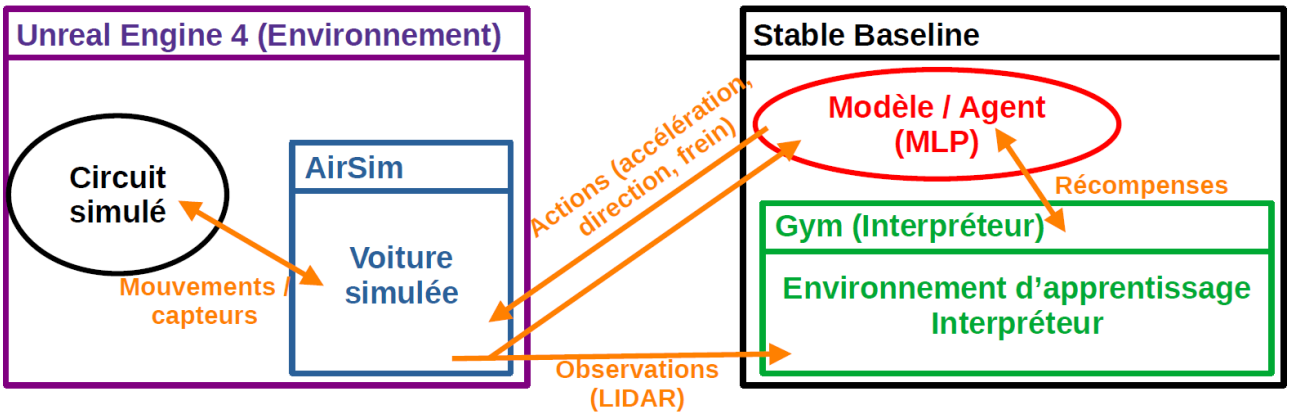

Figure 1 : Structure du problème et interaction des différents outils logiciels utilisés

Installation des outils

Pour ce projet, Python est utilisé avec les bibliothèques Gym et Stable baselines. Stable baselines est une bibliothèque Python permettant l'implémentation de nombreux algorithmes d'apprentissage par renforcement. De nombreux paramètres sont réglables lors de l'utilisation de Stable baselines, notamment la politique de l'agent, l'algorithme d'apprentissage par renforcement et ses hyperparamètres, ainsi que l'environnement d'apprentissage. Cet environnement d'apprentissage est défini à l'aide de la bibliothèque Python Gym.

Le moteur physique Unreal Engine est utilisé avec l'extension AirSim. Pour ce projet (et pour des raisons de compatibilité) nous utilisons le système d'exploitation Windows 10 ainsi que Unreal Engine 4.27 et Visual Studio 2019. Afin de procéder à l'installation des logiciels utilisés et des dépendances Python, veuillez suivre les informations disponibles sur le dépôt Github suivant : https://github.com/LudovicDeMatteis/Revue3EI_AirSim.

Le dossier contient plusieurs éléments importants ; le dossier *PythonClient/dqn_car* contient les scripts Python utilisés lors de ce projet le dossier *Unreal/Environments* contient deux environnements différents pouvant être exécutés en lançant le projet nommé *Blocks.uproject* ou

en lançant *Binaries/Win64/Blocks.exe* une fois l'environnement compilé par UE. Enfin, le fichier *settings.json* doit être déplacé dans un dossier nommé *AirSim* dans le dossier Documents de l'utilisateur Windows. Ce fichier contient plusieurs réglages de la simulation, notamment ceux des capteurs utilisés.

Plusieurs modèles pré-entrainés sont fournis et peuvent être directement utilisés, comme présentés dans les parties *'Démarrer l'apprentissage'* et *'Observer les résultats'* de cette ressource.

3 – Formalisme

3.1 - Commandes du véhicule via AirSim

Afin que l'agent (le véhicule autonome) puisse évoluer dans l'environnement simulé, il est nécessaire de définir les actions possibles. Dans le cadre d'un algorithme de Deep Q-Learning, que nous allons utiliser par la suite, l'espace des actions possible doit être discret. Nous allons donc chercher un espace des actions permettant de réaliser la tâche souhaitée.

Le simulateur AirSim permet de contrôler l'accélération du véhicule, la direction des roues et le freinage. L'accélération est commandée sous forme d'une grandeur scalaire comprise en 0 et 1, la direction est comprise entre -0,5 et 0,5 et le freinage est une grandeur binaire commandant l'appui ou non sur le frein.

Question 2 :

Proposez et justifiez un espace des actions permettant de répondre efficacement au problème (entraînement rapide, généralisable…) ?

Réponse 2 :

Il est important d'avoir suffisamment d'actions possible pour que le véhicule puisse parcourir l'intégralité du parcours et puisse s'adapter à de nouveaux circuits. Ainsi, il semble intéressant d'utiliser l'amplitude totale accessible. Il faut cependant sélectionner un nombre fini d'actions possibles et donc discrétiser l'espace continu proposé par AirSim.

Effectivement, un nombre trop faible d'actions ne laissera que peu de marge au véhicule pour s'adapter aux situations rencontrées. Par exemple, ne tourner qu'avec une valeur de braquage maximale à droite ou à gauche ne permet pas au véhicule d'adopter une conduite « optimale ». À l'inverse, un trop grand nombre de paramètres risque de ralentir l'apprentissage en obligeant le système à tester plus de combinaisons observations-actions.

Dans le cadre de l'activité pratique présentée ici, nous utiliserons 3 valeurs d'accélération possibles (0, 0.5 et 1) et 5 valeurs de directions (-0.5, -0.25, 0, 0.25, 0.5). Ceci nous donne donc 15 actions possibles, représentées en figure 2, auxquelles s'ajoute une action de freinage sans braquage.

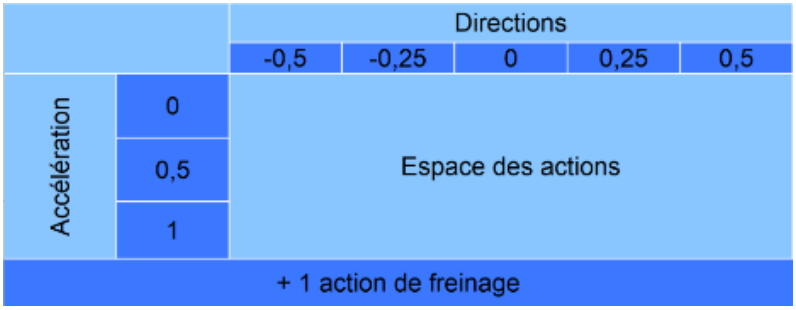

Figure 2 : Espace des actions utilisé pour la résolution de notre problème

La démarche présentée dans cette ressource reste valable pour un espace des actions différent. Il reste cependant important de garder une certaine symétrie dans la définition des directions afin d'éviter de biaiser ici la conduite vers la droite ou la gauche.

3.2 – Retour d'informations de l'environnement

Pour prendre une décision, l'agent a besoin d'avoir à disposition des informations sur son environnement. Ces informations, appelées observations, prennent la forme de données de capteurs. Pour le problème considéré, nous utiliserons un capteur LIDAR, comme mentionné précédemment. Il est cependant possible d'utiliser d'autres capteurs en changeant simplement le paramétrage du simulateur AirSim.

Le capteur que nous utilisons dans l'exemple présenté ici est paramétré avec 200 points par rotation et 10 rotations par secondes. Les faisceaux sont contenus dans un plan horizontal et avec un champ de vue de -120° à 120°. La figure 3 présente la voiture dans le simulateur AirSim ainsi que les points repérés par le capteur utilisé.

Figure 3 : Véhicule simulé et représentation des points détectés par le capteur LIDAR installé

Il est alors possible de récupérer directement dans un script Python les données du capteur. Une liste de 200 éléments est alors créée, contenant la position (x,y,z) de l'élément repéré par le LIDAR. La distance à l'obstacle ainsi que son angle associé sont obtenus à l'aide d'une conversion trigonométrique.

3.3 - Récompenses

La mise à jour de la politique de l'agent se fait par l'utilisation de récompenses. Une grande récompense indique que l'action choisie compte tenu des observations est bonne. À l'inverse, une récompense faible, voire négative, indique à l'agent que son choix n'était pas bon. L'apprentissage par renforcement optimise la politique pour maximiser la récompense totale. Celle-ci a donc un rôle central dans l'apprentissage. Dans notre cas, la politique étant un réseau de neurones dense (MLP), la récompense reçue est utilisée sous la forme d'une fonction coût dans l'algorithme de rétropropagation.

Il est important de proposer des récompenses qui pousse l'agent vers le comportement désiré.

Question 3 :

Présentez quelques aspects du comportement souhaité du véhicule. Comment traduire ces comportements sous forme de récompenses ?

Réponse 3 :

Afin de progresser sur le circuit, il faut tout d'abord que le véhicule avance. Ce comportement peut être vu de plusieurs manières ; cela peut correspondre à un déplacement du véhicule entre deux instants ou à une vitesse non nulle du véhicule. Dans le premier cas, la position du véhicule peut être enregistrée à chaque instant et la distance euclidienne entre deux positions successives est choisie comme récompense R_p. Le second point de vue se traduit en utilisant simplement une fonction de la vitesse comme récompense R_v.

Il faut également que la voiture reste éloignée des murs afin qu'elle progresse dans le circuit sans collisions. Une récompense possible pour favoriser ce comportement est la distance minimale à un obstacle. En effet, pour maximiser cette récompense, l'agent devra se trouver au milieu de la route. Cette récompense R_d est alors définie dans notre cas comme la valeur minimale contenue dans la liste des observations issues du LIDAR.

La récompense totale est alors définie comme $R = \alpha R_p + \beta R_d$ ou $R = \alpha R_v + \beta R_d$ avec α et β des paramètres réglables permettant d'ajuster les ordres de grandeurs dans différents éléments et d'accentuer l'un ou l'autre des comportements.

D'autres comportements peuvent être pensés et ajoutés sous forme de récompense, comme favoriser une trajectoire avec moins de virages (limitant l'oscillation du véhicule en ligne droite) ou mesurer l'avancée du véhicule par la traversée de points de contrôles.

4 – Définir les éléments d'apprentissage

4.1 - Définition d'un environnement Gym

Maintenant que les éléments principaux du problème sont définis (observations, simulation, récompenses), il est possible de mettre en place l'environnement d'apprentissage qui nous permettra d'entrainer l'agent. Pour cela, nous avons défini un environnement personnalisé en utilisant la bibliothèque Python Gym.

Pour permettre le fonctionnement de l'environnement d'apprentissage, il faut implémenter plusieurs méthodes particulières :

- *__init__* : cette méthode, constructeur de l'environnement d'apprentissage, **initialise l'agent** et les grandeurs qu'il utilisera, notamment l'espace des actions.

- *step* : cette méthode réalise une action de l'agent. Prenant en argument une action, elle **applique l'action**, c'est à dire la réalise en simulation, puis **récupère les observations** et **calcule les récompenses** qui seront transmises à l'algorithme d'apprentissage pour mettre à jour la politique.

- *reset* : cette méthode **replace la voiture dans son état initial**. Afin de limiter les biais lors de l'apprentissage, la voiture a plusieurs points d'apparitions possibles. Le choix du lieu de réapparition est aléatoire.

À travers les différentes méthodes définies dans cet environnement, l'agent interagit avec l'environnement simulé. Cette interaction est possible grâce à l'utilisation de la bibliothèque fournie par AirSim, permettant la récupération des données des capteurs et l'envoi de commandes au véhicule.

4.2 - Définir l'agent

Le dernier élément à définir pour permettre l'apprentissage est la politique de l'agent et l'algorithme d'apprentissage associé. Dans l'exemple fourni, et comme précisé précédemment, la politique de l'agent se base sur l'utilisation d'un réseau de neurones totalement connecté (MLP).

L'algorithme d'apprentissage utilisé est un algorithme de double deep Q-Learning (DQN).

Ces éléments sont définis à l'aide de la bibliothèque Python Stable Baselines et les hyperparamètres sont gardés à des valeurs par défaut pour la plupart. La définition du modèle et des valeurs des hyperparamètres ayant changé sont présentés en figure 4.

Il est intéressant d'observer les effets des différents hyperparamètres et changeant les paramètres avant l'apprentissage. En particulier, l'effet du taux d'apprentissage (*learning_rate*) et du taux d'exploration (*exploration_rate*) sont facilement observables. Dans l'exemple donné, le taux d'exploration est gardé par défaut et n'apparait donc pas dans la définition du modèle.

```
model = DQN(
    "MlpPolicy",
    env,
    learning_rate=0.01,
    verbose=1,
    batch_size=64,
    train_freq=16,
    target_update_interval=2000,
    learning_starts=100,
    buffer_size=10000,
    tensorboard_log="./tb_logs/",
)
```

Figure 4 : Définition du modèle et de l'algorithme d'apprentissage.
L'environnement Gym défini est associé à ce modèle.

4.3 - Démarrer l'apprentissage

Nous pouvons alors démarrer la phase d'apprentissage. Pour cela, il est nécessaire de démarrer la simulation, comme présenté lors de l'installation d'Unreal Engine et d'AirSim.

Le script *dqn_car.py* est ensuite exécuté en s'assurant que la variable *load* est initialisée à *False*. Un nouveau modèle est alors créé et la phase d'apprentissage est lancée pour un nombre d'étape définie, comme présenté figure 5. Les meilleurs modèles sont régulièrement enregistrés et pourront ensuite être visualisés.

```
model.learn(
    total_timesteps=100000,
    tb_log_name="dqn_airsim_car_run_" + str(time.time()), **kwargs
)
```

Figure 5 : Définition de la phase d'apprentissage sur 100 000 étapes.
Les performances sont enregistrées à l'aide de tensorboard.

4.4 - Optimiser l'apprentissage

Afin d'accélérer l'apprentissage, il est possible d'accélérer le temps en simulation. Cela permet de réduire fortement le temps nécessaire pour entrainer un modèle en le réduisant à moins de deux heures (en vitesse x5). Pour cela, dans le fichier de configuration AirSim, placé dans les documents utilisateurs, permet de régler le paramètre « *ClockSpeed* » qui est pris en compte lors du lancement

de la simulation. La même valeur doit être précisée au début du fichier Python *envs/car_env.py* avant le début de l'entrainement du modèle. Il est important de noter que le modèle n'est pas impacté directement par l'accélération du temps. Il faudra cependant régler en conséquence le nombre d'actions par secondes.

Le réglage de la variable *load* a True permet de charger un modèle déjà partiellement entrainé et de continuer l'apprentissage. Cette option est particulièrement utile lors de l'utilisation de ce système en séance d'activité pratique, permettant le pré-entraînement d'un modèle avant la séance puis la finalisation de l'entrainement en 15 minutes durant la séance. Si le modèle initial n'est pas trop entrainé, les performances seront significativement améliorées lors de cette deuxième phase, permettant alors aux élèves d'observer simplement l'évolution du comportement de l'agent. Entre les deux phases d'apprentissage, il est également possible de modifier la définition des récompenses et d'observer alors rapidement leur effet. En revanche, un changement des hyperparamètres, de l'algorithme d'apprentissage ou de la politique (tous définis lors de l'initialisation du modèle), demande de créer un nouveau modèle qui doit reprendre l'entraînement depuis le début.

5 – Observer les résultats

5.1 - Utilisation d'un circuit de test

L'évolution de la voiture sur le circuit peut être visualisée au cours de l'entrainement. Cela permet d'avoir un premier aperçu des performances de la voiture mais cela n'est pas suffisant.

Question 4 :

L'observation des performances de la voiture sur le circuit d'entrainement sont-elles suffisantes ? Quel est l'intérêt d'utiliser un autre circuit pour tester le modèle ?

Réponse 4 :

La visualisation des performances sur le circuit d'entrainement n'est pas représentative des performances réelles de l'agent. Nous pouvons être dans le cas du problème de surapprentissage, ayant alors un véhicule très performant sur des données connues (le circuit d'entrainement) mais échouant sur des données nouvelles (un nouveau circuit). Il est alors important d'utiliser un **circuit de test,** sur lequel le véhicule ne s'est pas entrainé, pour évaluer les performances de l'agent.

Les circuits utilisés pour notre exemple, et fournis comme exemples, sont présentés figure 6.

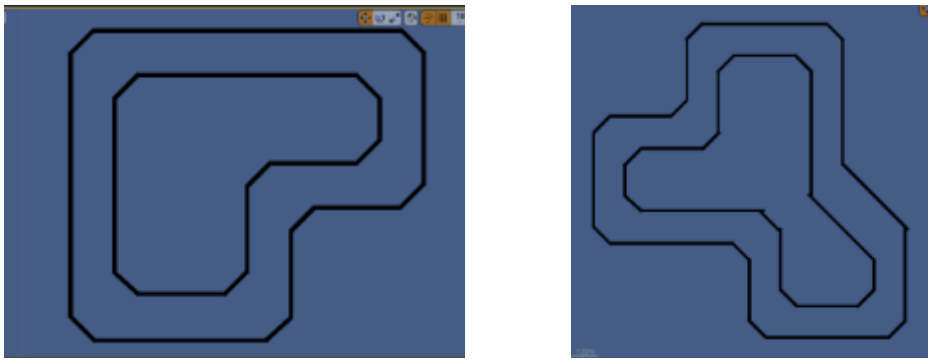

Figure 6 : Circuits utilisés : circuit d'entraînement à gauche et de test à droite.
Les deux circuits doivent être différents pour évaluer la capacité de généralisation de l'agent mais doivent tout de même garder des caractéristiques similaires.

Afin d'observer l'évolution de l'agent sur l'un des circuits, il faut lancer la simulation du circuit correspondant puis lancer le script *dqn_car_test.py*. Dans ce dernier, il est nécessaire de préciser quel modèle utiliser et donc mettre le chemin d'accès au modèle enregistré lors de l'entrainement.

Ce script commence par réaliser plusieurs passages sur le circuit afin d'obtenir la récompense moyenne associée au modèle. Le véhicule se déplace en simulation ce qui donne également un aperçu des performances de l'agent.

Une vidéo présentant la conduite sur le circuit de test par un modèle pré-entraîné fourni est présentée sur le site Culture Science de l'Ingénieur et accessible par ce lien [7].

4.2 - Visualiser une trajectoire pour un modèle

Le même script affiche ensuite la trajectoire de la voiture afin de donner un aspect graphique permettant d'évaluer les performances. La figure 7 présente un exemple de ces trajectoires pour un modèle entrainé en un peu moins de deux heures.

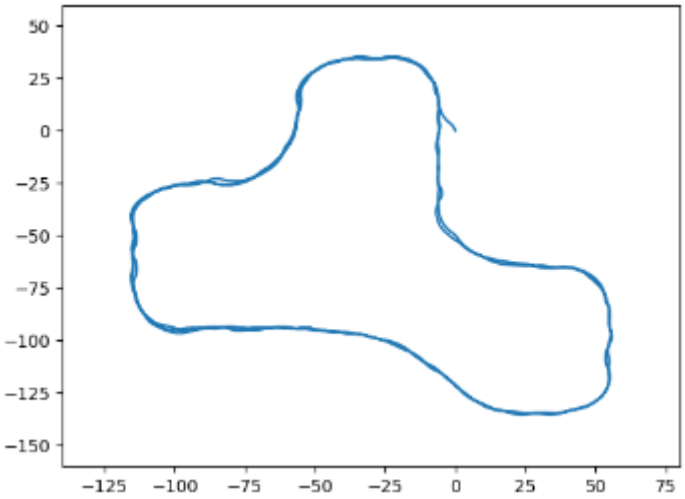

Figure 7 : Exemple d'une trajectoire obtenue sur le circuit de test pour un agent entrainé pendant environ 2h

La comparaison des trajectoires à différentes phases de l'entrainement permet d'observer simplement l'apport de l'apprentissage par renforcement dans le cadre de la conduite autonome.

6 – Conclusion

Dans cette ressource, nous avons présenté un cas d'application concret d'un problème d'apprentissage par renforcement. Cette activité illustre l'apprentissage de la conduite autonome en comparant des agents avant et après la phase d'entrainement. L'effet des hyperparamètres peut également être étudié en changeant leur valeur dans la définition du modèle. Des modèles pré-entrainés permettent d'accélérer la phase d'apprentissage et de permettre ainsi de réaliser cette étude lors d'une séance de travaux pratiques.

Les méthodes présentées dans cet article sont adaptables simplement à d'autres problèmes d'apprentissage par renforcement, si l'on dispose d'un environnement simulé. Il est encouragé d'appliquer ces méthodes à de nouveaux problèmes afin d'observer le vaste champ d'application de l'apprentissage par renforcement.

Références :

[1]: Documentation Stable Baselines 3 et Gym, https://stable-baselines3.readthedocs.io/en/master/, https://www.gymlibrary.ml/

[2]: Documentation AirSim, https://microsoft.github.io/AirSim/

[3]: V. Mnih et al., Playing Atari with Deep Reinforcement Learning, arXiv, https://arxiv.org/abs/1312.5602

[4]: V. Mnih et al., Human-level control through deep reinforcement learning, *Nature* **518,** 529-533 (2015), https://www.nature.com/articles/nature14236

[5]: Dossier Intelligence Artificielle, juin 2022, https://eduscol.education.fr/sti/si-ens-paris-saclay/ressources_pedagogiques/dossier-intelligence-artificielle

[6]: Introduction à l'apprentissage profond, S. Janny, L. De Matteïs, W. Shu-Quartier, https://eduscol.education.fr/sti/si-ens-paris-saclay/ressources_pedagogiques/introduction-lapprentissage-profond

[7]: Apprentissage par renforcement de la conduite d'un véhicule au AirSim, L. De Matteïs, S. Radosavljevic, juillet 2022, https://eduscol.education.fr/sti/si-ens-paris-saclay/ressources_pedagogiques/apprentissage-par-renforcement-dela-conduite-dun-vehicule-sur-airsim