

ENSEIRB-MATMECA

TÉLÉCOMMUNICATIONS - I2SC



Rapport de stage de fin d'études

Développement d'un algorithme d'apprentissage par renforcement
pour la prise de décision temps réel

Lucas STEFFANUTTO

Tuteur :

Mohamad ALBILANI

Table des matières

1 Remerciements	3
2 Résumé	3
3 Abstract	3
4 Introduction	4
5 Présentation de l'entreprise	5
5.1 Advans Group	5
5.2 Avisto	5
5.3 Recherche et Développement (R&D)	6
6 Projet KARlab	6
6.1 Présentation	6
6.2 Contexte Technique	7
6.2.1 Architecture et technologies utilisées	7
6.2.2 Les systèmes embarqués sur le kart	8
6.3 Use Cases	9
6.3.1 Use Case 1 : Mode d'entraînement d'un pilote	9
6.3.2 Use Case 2 : Mode de détection des dangers	9
6.3.3 Use Case 3 : Mode de démonstration de stationnement automatique	10
6.3.4 Use Case 4 : Mode de jeu	10
6.4 Organisation	10
6.4.1 Organisation du projet	10
6.4.2 Organisation de mon stage	11
7 Introduction à l'Apprentissage par Renforcement	12
7.1 Description	12
7.2 Modélisation	13
7.3 Résolution	13
7.3.1 Fonction de valeur d'état	13
7.3.2 Fonction de valeur d'état-action	14
7.3.3 Optimalité	14
7.4 Résolution et Programmation Dynamique	15
7.4.1 Évaluation de la politique	15
7.4.2 Amélioration de la politique	16
8 Algorithmes classiques	17
8.1 Approches basées politique	17
8.1.1 Monte Carlo Learning	18
8.1.2 Monte Carlo Policy Gradient	18
8.2 Approches basées valeur	20
8.2.1 Temporal Difference Learning (TD-Learning)	20
8.2.2 Q-Learning	20
8.2.3 Deep Q-Learning	21
8.3 Approches mixtes	21
8.3.1 Actor-Critic	21
8.3.2 Asynchronous Advantage Actor-Critic (A3C)	22

9 Algorithmes État de l'Art	24
9.1 Proximal Policy Optimization (PPO)	24
9.2 Autres algorithmes états de l'art et limites des méthodes d'AR	26
9.2.1 Deep Deterministic Gradient Policy (DDPG)	26
9.2.2 Soft Actor Critic (SAC)	27
9.2.3 Limites des techniques d'AR	28
10 État de l'art du stationnement autonome par AR : Analyse et choix technique	29
10.1 Analyse de l'existant	29
10.2 Discussion sur les papiers étudiés et sur les algorithmes d'AR	32
10.2.1 Limites des papiers de l'état de l'art	33
10.2.2 Similitudes des papiers de l'état de l'art	33
10.3 Solution envisagée	34
11 Réalisations	35
11.1 Modélisation du stationnement autonome par AR sur Unity	35
11.1.1 Modélisation	35
11.1.2 Unity ML-Agents	36
11.1.3 Entraînements et tests	37
11.1.4 Résultats	44
11.2 Modélisation du stationnement autonome par AR sur Carla simulator	46
11.2.1 Le simulateur	46
11.2.2 Mise en place du modèle d'AR sur CARLA	46
12 Retour sur expérience	48
13 Conclusion	49
14 Bibliographie	50
15 Annexes	53

1 Remerciements

Je tiens à remercier Mohamad Albilani, mon maître de stage durant ces 6 mois, pour sa disponibilité et son aide précieuse. Je remercie également Hélène Fiotti et Jean-François Beraud pour m'avoir fait confiance et ainsi permis d'intégrer Avisto, pour y effectuer mon stage dans le domaine que je souhaitais, sur un sujet aussi intéressant qu'instructif et formateur. Je remercie les personnes que j'ai eu la chance de côtoyer au quotidien au sein du projet KARlab, qui ont permis mon intégration et une excellente cohésion de groupe. Nos échanges interdisciplinaires ont été bénéfiques et j'ai beaucoup appris d'eux. Enfin, je remercie l'équipe pédagogique de l'ENSEIRB-MATMECA qui m'a suivie durant ces trois années grâce auxquelles j'ai énormément évolué, sur l'aspect technique mais également méthodologique et relationnel d'un ingénieur.

2 Résumé

Ce document retrace le stage réalisé par Lucas STEFFANUTTO sur le projet KARlab d'ADVANS Group. Ce projet représente une vitrine technologique pour l'entreprise. Son but est de réaliser un mix entre kart autonome et course multi-joueurs en réalité mixte, inspirée de jeu de karting.

Le sujet de ce stage concerne l'apprentissage par renforcement du stationnement autonome pour un kart. L'objectif principal de ce stage est de réaliser l'ébauche de la partie prise de décision en temps réel du kart. Cette supervision est un cerveau virtuel qui va devoir apprendre un comportement complexe et remplacer un pilote réel : il réalisera les accélérations, freinages et changements de directions à sa place, pour rejoindre sa place de parking. Pour se faire, le kart perçoit son environnement grâce à des capteurs et apprend par le biais de l'Apprentissage par Renforcement (AR).

Après avoir réalisé une étude documentaire et étudié les différentes solutions existantes expérimentées pour le stationnement autonome, la première étape a été de définir un modèle d'AR, et de le modéliser sur Unity avec le plugin ML-Agents. Plusieurs Proof of Concept (PoC) ont été réalisées, et un modèle de réseau de neurones a été obtenu et représente le comportement optimal final. Le transfert de l'apprentissage, sur le simulateur CARLA, plus proche de la réalité, est en cours.

Ce bloc de supervision est appelé à évoluer et à être affiné dans le temps pour son utilisation embarquée sur le kart.

3 Abstract

This document describes the internship achieved by Lucas STEFFANUTTO on the ADVANS Group KARlab project. It represents a technological showcase for the company. Its goal is to create a mix between autonomous kart and a mixed reality multi-player race, inspired by karting game.

The subject of this internship concerns the reinforcement learning of autonomous parking for a kart. The main objective of this internship was to carry out the draft of the real-time decision making part of the kart. This supervision part is a virtual brain that will have to learn a complex behavior and replace a real driver : it will carry out the accelerations, braking and changes of direction in its place, to reach its parking place. To do so, the kart perceives its environment thanks to sensors and learns through Reinforcement Learning (RL).

After having conducted a literature review and studied the different existing solutions experimented for autonomous parking, the first step was to define a RL model, and create it on Unity with the ML-Agents plugin. Several Proof of Concept (PoC) were performed, and a neural network model was obtained and represents the final optimal behavior. The transfer of the learning, on the CARLA simulator, closer to reality, is in progress.

This supervision block will evolve and be refined in time for its embedded use on the kart.

4 Introduction

Le nombre de voitures dans le monde croît avec l'augmentation de la population mondiale. Se garer en ville est une problématique qui s'intensifie avec l'étalement urbain. La rareté des places de parking et la condensation des espaces est une des causes du développement de systèmes d'aide à la conduite automobile (Advanced Driver-Assistance Systems ou ADAS).

Parmi eux se trouvent les aides au stationnement, comme la présence de capteurs sur le véhicule, pour aider l'utilisateur durant ses manœuvres (radar, caméras, etc.). Le stationnement mains libres permet au véhicule de contrôler la direction en laissant au conducteur le choix du freinage et de l'accélération. Dans le cas du stationnement automatique, l'automobile prend le contrôle total des commandes pour effectuer la manœuvre de parking (Automated Parking System ou APS). Ces modules d'assistance permettent de garantir confort, sécurité et gain de temps à l'utilisateur. L'utilisation des parkings est ainsi optimisée.

Les systèmes classiques d'APS réalisent une planification puis un suivi de trajectoire. Néanmoins, ils peuvent être sujet à un manque de précision, et la non-linéarité de la dynamique du véhicule peut causer une déviation de l'itinéraire initial. De ce fait, l'étude et l'utilisation de nouvelles solutions de stationnement autonome peuvent être intéressantes. C'est dans ce contexte que l'on peut introduire l'apprentissage par renforcement, où un algorithme apprend à un agent (un kart dans notre cas) à se garer correctement, par la pratique de multiples expériences. Le modèle développé peut alors associer une action optimale, soit les commandes du véhicule à actionner, en fonction de chacun des états possibles dans lesquels il peut se retrouver.

Cette problématique s'inscrit dans le cadre de mon stage : Avisto m'a accueilli au sein du projet KARlab, dans le but de modéliser et de développer une fonctionnalité de stationnement autonome par apprentissage par renforcement. Dans le futur, elle sera embarquée sur le kart.

Les missions qui m'ont été confiées sont diverses : La réalisation de l'état de l'art de l'Apprentissage par Renforcement (AR), ainsi que celui du stationnement autonome à partir de cette technique. La modélisation et la mise en place de la problématique d'APS pour le kart de KARlab, puis la réalisation d'une première PoC de stationnement autonome par AR en simulation sur Unity. Une deuxième, se rapprochant beaucoup plus de la réalité, sur CARLA Simulator est aussi en cours de développement. La documentation technique associée aux recherches et travaux effectués a aussi été réalisée. Durant ce stage j'ai aussi participé aux différentes démonstrations du produit devant des responsables d'affaire et des responsables techniques.

Le rapport se divise en plusieurs parties avec dans un premier temps une description de l'entreprise, suivie de celle du projet KARlab et de son organisation. Les bases de l'AR et son état de l'art sont abordés. L'état de l'art du stationnement automatique par AR est aussi résumé. Une description de la solution envisagée et proposée est expliquée par la suite. Pour finir, un retour sur expérience et une conclusion marquent la fin du rapport.

5 Présentation de l'entreprise

5.1 Advans Group

ADVANS Group est un ensemble de sociétés d'ingénieries spécialisées dans la conception de systèmes électroniques, logiciels et mécaniques. La mission d'ADVANS Group est d'imaginer et de concevoir aujourd'hui les systèmes de demain, aux côtés des acteurs majeurs des secteurs où ils sont utilisés. Les filiales de ce groupe accompagnent de grandes sociétés internationales, des ETI, des PME et des start-up. Le groupe intervient dans de nombreux domaines : aérospatiale, automobile, défense, énergie, IoT, médical, secteur tertiaire, semi-conducteur, transport etc. Cette Société par Action Simplifiées (SAS) propose de nombreux services à ses clients par le biais de ses ingénieurs : Assistance technique ponctuelle et consulting, déploiement d'équipes techniques pour l'élaboration de produit fini, mais aussi maintenance et validité des systèmes. Des services de formation et d'externalisation en Serbie sont aussi possibles. Fondé et managé par des ingénieurs, ADVANS Group cultive un modèle spécifique avec une identité et des valeurs fortes. La société emploie plus d'un millier d'ingénieurs au sein de trois entreprises : ELSYS Design (systèmes embarqués), AVISTO (développement logiciel) et MECAGINE (mécanique). La Figure 1 ci-dessous illustre l'organisation du groupe.



FIGURE 1 – Organisation et implantation d'ADVANS Group

ADVANS Group est aussi présent dans 7 pays : Les équivalents Serbes d'Avisto et d'Elsys Design sont présent dans l'Est de l'europe, et une filiale d'Elsys Design est présente au sein de la Silicon Valley.

5.2 Avisto

AVISTO est une SAS créée en 2004 par M. Radomir JOVANOVIC. Son siège social se situe à Vallauris (06220) et ses implantations sont situées dans les mêmes villes que celles de la Figure 1. C'est une société d'ingénierie dont la vocation est le développement de solutions logicielles. Elle se spécialise dans la réalisation de projets complexes mettant en œuvre des technologies "Objet" (Java/JEE, C++, C# et .Net, Core, Python, PHP), Web (Angular, ReactJS, VueJS) et Mobile (Android, iOS, multiplateforme). Son effectif est compris entre 200 et 249 salariés. Sur l'année 2019 elle réalise un chiffre d'affaires de 17 659 300,00 €. Le total du bilan a augmenté de 14,30% entre 2018 et 2019.

AVISTO couvre toutes les étapes de la vie d'un logiciel, du recueil des besoins à la préparation de ses évolutions futures. A ce cœur de compétences, AVISTO a progressivement greffé des métiers périphériques au développement logiciel, notamment le DevOps, l'assurance qualité, la data science et la cybersécurité. La compagnie peut aussi couvrir les systèmes d'information, le web, les télécoms, le logiciel industriel, l'applicatif embarqué et le réseau.

Pour le management de projet, les ingénieurs d'AVISTO suivent tantôt des méthodes Agile, tantôt le cycle en V. Comme décrit pour ADVANS Group, pour réaliser ses développements, AVISTO propose des modes

d'intervention variés (Assistance Technique, Forfait, Centre de Services...), en interne ou sur site client, avec différents niveaux d'engagement. Des développements en mode nearshore par l'intermédiaire de la filiale AVISTO Eastern Europe, située en Serbie sont également possibles.

5.3 Recherche et Développement (R&D)

ADVANS Group et ses filiales ont choisi de se focaliser sur leur cœur de compétences, à savoir la technique et la technologie. Elles ont entrepris des partenariats avec trois jeunes entreprises à fort potentiel : Wildmoka, solution cloud d'édition vidéo pour créer, distribuer, et monétiser des contenus sur le digital ; Upmem, processeurs innovants pouvant être directement intégrés dans la mémoire vive des serveurs pour révolutionner l'efficacité des calculs dans les data centers ; LucieLabs, plateforme technologique permettant de contrôler l'éclairage à haute intensité. ADVANS Group collabore également avec Newron Motors, qui a annoncé début 2020 le lancement des pré-commandes de sa moto électrique EV-1. ADVANS Lab, une structure d'accompagnement et d'incubation mise en œuvre en 2019, profitent par ailleurs à ce jour à d'anciens collaborateurs d'ELSYS Design.

Les activités de R&D et d'innovation peuvent donc se faire sous forme de projets en collaborations avec des clients : Projet d'Intelligence artificielle embarquée sur des nano-satellite avec l'Agence Spatiale Européenne (ESA), ou encore projet d'interactions entre le conducteur et le véhicule semi-autonome avec l'Université Côte d'Azur et Renault Software Labs sur le projet ADAVEC.

La R&D interne a pour but principal de faire monter en compétences les ingénieurs du groupe sur des problématiques susceptibles d'intéresser ses clients. Actuellement, le projet phare est KARlab, qui applique la réalité mixte à un environnement de conduite semi-autonome.

6 Projet KARlab

6.1 Présentation

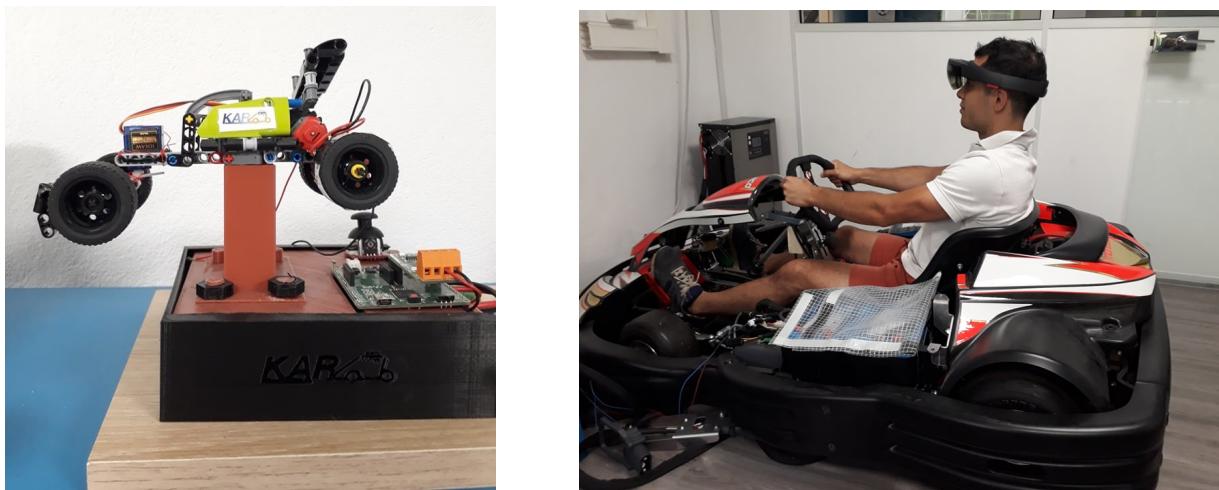


FIGURE 2 – Projet KARlab : état initial et état actuel

KARlab est un projet de R&D multi-disciplinaire mené conjointement depuis 2018 par ELSYS Design, Mécagine et AVISTO. Il consiste à réaliser un kart constitué de systèmes intelligents pour initier les conducteurs aux interactions avec les véhicules de demain. Ce projet développe les savoir-faire technologiques du groupe ADVANS aux niveaux électronique (systèmes embarqués) et logiciel (technologies immersives, IA, Big data...) appliqués aux métiers de l'automobile. Le projet propose également une course de karting en réalité augmentée et virtuelle à l'utilisateur. Muni d'un casque de réalité mixte (Microsoft Hololens figure 2 à droite), le pilote interagit avec des éléments virtuels placés dans son champs de vision dans le monde réel. Dans le même esprit que *Mario kart*, les interactions virtuelles entraînent des évènements dans le monde réel lors de la course : actions

sur le kart du joueur ou sur celui de ses adversaires (accélérations, ralentissements...). La figure 2 permet de visualiser les différentes évolutions du projet, de sa première Proof Of Concept (PoC), à gauche, jusqu'à son état actuel où les travaux sont réalisés avec un vrai kart, droite.

6.2 Contexte Technique

6.2.1 Architecture et technologies utilisées

Dans ce projet, 3 entités communiquent entre elles : le kart, le serveur de jeu et l'hololens. La communication est réalisée en WIFI via le protocole TCP. Ces différents blocs peuvent s'envoyer des messages "*FlatBuffers*" entre eux. Les figures 3 et 4 résument les interactions entre les entités.

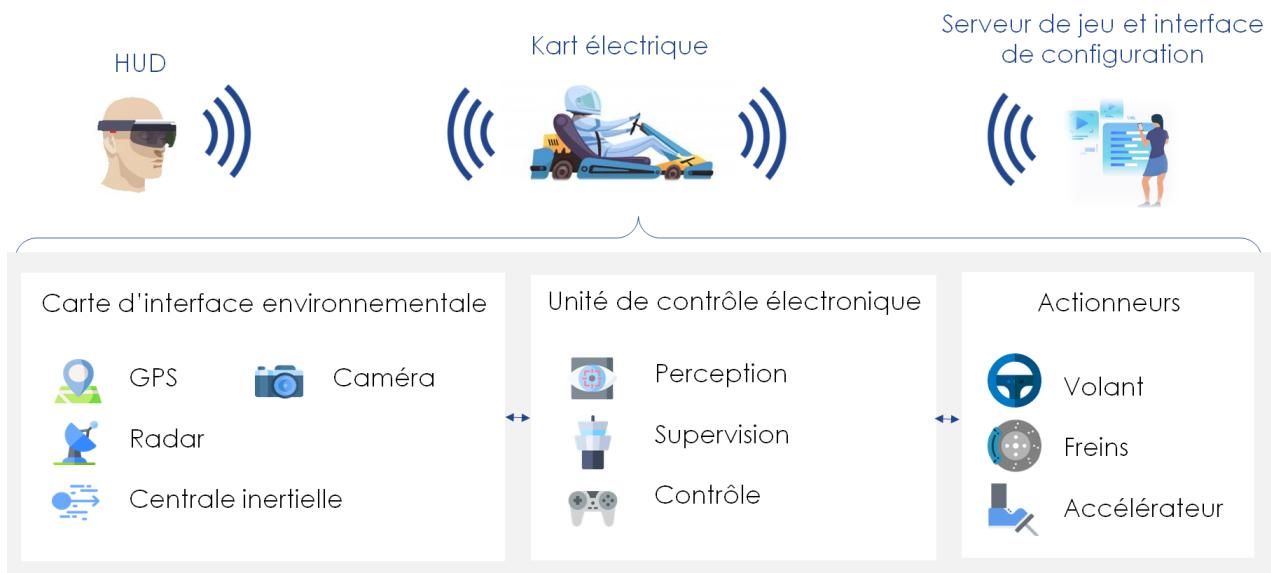


FIGURE 3 – Schéma résumant les communications entre les entités du projet KARlab, ainsi que les modules présents à bord du kart

Les systèmes embarqués à l'intérieur du kart illustrés Figure 5, se répartissent les tâches de la manière suivante, comme schématisé figure 3 : La première partie concerne la Carte d'Interface Environnementale (CIE) et permet au kart de percevoir les informations relatives à son environnement (sa position, sa vitesse etc.) apr le biais de capteurs, 5 (droite) tels qu'un radar, un GPS ou une centrale inertuelle (IMU). La CIE va échanger avec l'Unité de Contrôle Electronique (UCE), schématisé à gauche et illustrée à droite figure 5. L'UCE va synthétiser les données recueillies, analyser l'information et superviser le contrôle qui doit être effectué sur le kart. La dernière partie mécanique est composée des Actionneurs qui agiront sur le kart au travers de l'angle des roues, de l'accélération et du frein.

Le kart communique avec un Hololens, porté par le pilote : L'utilisateur pourra apercevoir dans son champs de vision les notifications liées au kart en temps réel, comme la détection d'un panneau, ou bien encore sa vitesse.

Enfin, un serveur central permet de coordonner le jeu. Maître du scénario, il communique les informations avec l'ensemble des karts sur la piste. Les karts sont également inter-connectés, figure 4 et pourront échanger des données directement entre eux.

Les technologies utilisées à bord du kart fonctionnent et communiquent essentiellement en langages C et C++. Les parties supervisions et intelligences artificielles sont d'abord développées et testées en Python ou en C#, sur simulateurs, avant d'être embarquées. Les technologies utilisées pour le serveur et l'Interface Homme Machine (IHM) sont les langages React et JavaScript.

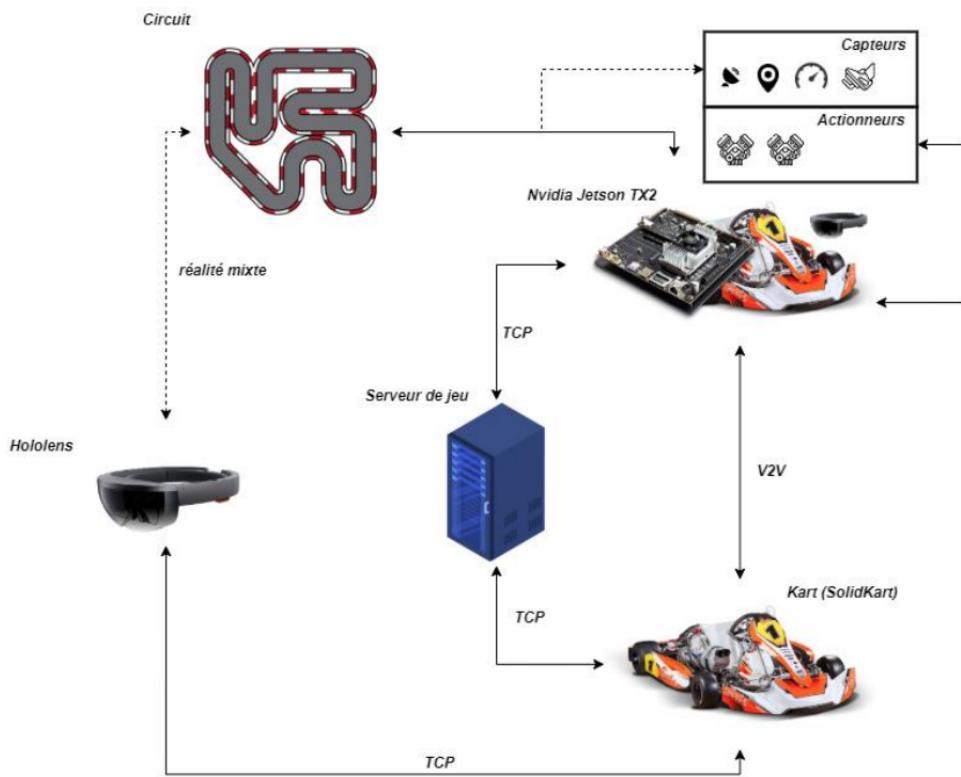


FIGURE 4 – Schéma des communications entre les entités KARlab

6.2.2 Les systèmes embarqués sur le kart

Pour ce projet, un kart électrique classique *Sodikart* est utilisé. Il est combiné à une carte électronique embarquée *Nvidia Jetson TX2*, jouant le rôle de l'UCE décrite précédemment. Celle-ci joue le rôle de carte mère pour le système : elle traite les informations reçues et permet de gérer les évènements importants. Durant la conception du projet, il a été décidé que deux systèmes d'exploitation seraient utilisés simultanément : un RTOS (Real Time Operating System), nommé Erika, et un Rich OS (Linux) classique. Le premier est en charge de gérer les décisions critiques dont peuvent dépendre l'intégrité du kart et la sécurité du pilote (freinage d'urgence par exemple). Il est en charge d'envoyer les commandes aux actionneurs, figure 3. Les décisions critiques doivent être prise par un RTOS, de manière à gérer les évènements très rapidement, et se rapprocher le plus possible du temps réel. Le système Linux est en charge des données non-critiques et relatives au jeu. Il se connecte au serveur de jeu, récupère les données et les stocke en mémoire sur le kernel. Le RTOS se sert aussi de ces données pour prendre ses décisions. Le FreeRTOS figure 5 assure la liaison entre les CPUs et le bus CAN. L'hyperviseur se situe entre les OS et la partie hardware. Son rôle est de permettre à plusieurs OS de fonctionner en même temps sur une seule machine physique. Il attribue les composants matériels à chaque OS : par exemple, un accès au périphérique WIFI est attribué à la partie RTOS. L'hyperviseur répartit aussi la puissance de calcul présente sur la carte entre chaque OS. Cela permet d'optimiser la gestion des ressources à bord du kart. Les systèmes d'exploitation communiquent entre eux via un espace mémoire partagé.

La seconde carte embarquée à bord du kart est la **CIE**. Elle regroupe tous les capteurs présents à bord du kart : Radar, GPS, Centrale Inertielle (IMU). Cette carte recueille les données environnementales, les trie et les envoie à l'UCE. Ce dernier pourra effectuer le traitement associatif (Position et vitesse du kart, Détection de dangers, fusion de données, filtre de Kalman étendu) et contrôler les actionneurs mécaniques en conséquences.

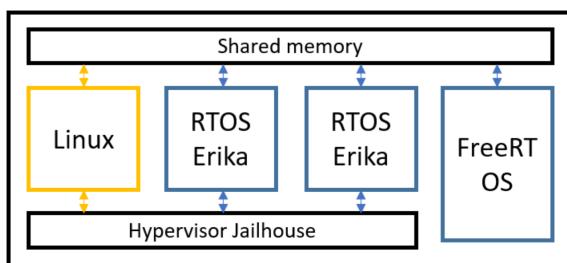


FIGURE 5 – Schéma de l'architecture de L'UCE (gauche) et photo des composants embarqués sur le kart à (droite).

6.3 Use Cases

Plusieurs spécificités sont en cours de développement sur le projet KARlab.

6.3.1 Use Case 1 : Mode d'entraînement d'un pilote

La première fonctionnalité du système KARlab correspond à un mode d'entraînement pour le pilote, qui est équipé d'un Hololens. Ce casque de réalité augmentée lui permet de percevoir des objets projetés sur son champ de vision. L'utilisateur doit pouvoir apercevoir la trajectoire idéale à suivre, qui s'affiche virtuellement sur la piste, en plus des zones où il est recommandé d'accélérer ou de freiner. Un exemple de ce mode d'utilisation est illustré à gauche figure 6. Plusieurs modes d'assistance sont également disponibles et le kart peut amorcer lui-même les actions de freinage ou d'accélération.

6.3.2 Use Case 2 : Mode de détection des dangers

Ce mode de fonctionnement permet au kart de reconnaître une situation dangereuse. Les dangers peuvent être détectés de deux façons. La première plutôt classique grâce à des capteurs de type LIDAR, RADAR. La caméra et l'algorithme Yolov3 embarqué, permettent aussi de détecter les panneaux de signalisation et les potentiels obstacles, figure 6 à droite. De cette manière le kart peut adapter son comportement en réduisant sa vitesse ou en effectuant un freinage d'urgence. L'hololens peut aussi être utilisé pour visualiser en temps réel ce que le kart détecte et pour communiquer directement une alerte et les actions à réaliser en conséquence au pilote.



FIGURE 6 – Exemple d'illustrations du Use Case 1 entraînement d'un pilote (gauche), Inférence de l'algorithme Yolov3 pour le Use Case 2 détection de dangers (droite)

6.3.3 Use Case 3 : Mode de démonstration de stationnement automatique

Ce mode doit permettre au kart de réaliser un stationnement automatique : il doit regagner la place de parking qui lui est associée, dont la position est connue. La tâche doit s'effectuer de manière autonome, et sans pilote à l'intérieur du kart. Le véhicule n'est pas soumis à une limite de temps et doit emprunter le chemin le plus simple en marche avant, en gérant les éventuelles obstacles qui peuvent apparaître (piéton traversant le circuit, obstacle, éventuellement d'autres karts etc.). Ce système est développé en utilisant l'Apprentissage par Renforcement. Cette fonctionnalité sur laquelle j'ai personnellement travaillée concerne le plus haut niveau de conduite autonome, numéro 5, figure 7 dessous. La conduite autonome est divisée en 3 partie, à savoir, la perception de l'environnement, la prise de décision et le contrôle du véhicule. Durant ce stage c'est la partie de prise de décision qui a été développée.

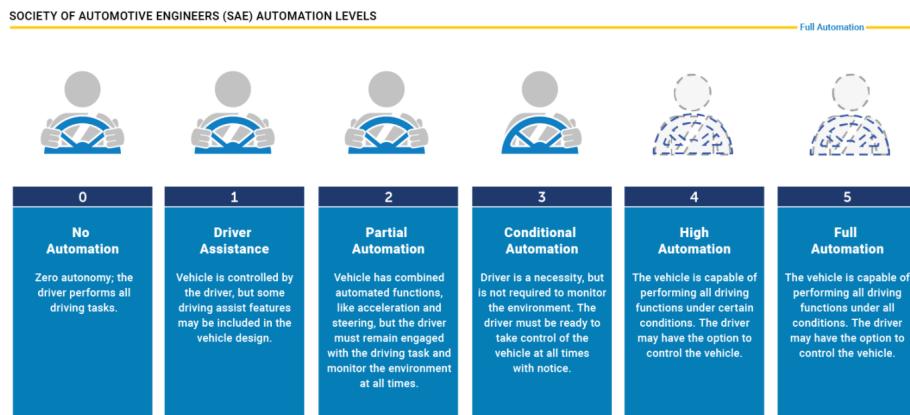


FIGURE 7 – Schéma des degrés d'autonomie d'un véhicule, d'après SAE International

6.3.4 Use Case 4 : Mode de jeu

Si les trois premières fonctionnalités traitent de problématiques classiques de véhicule autonome, le Use Case 4 permet à plusieurs joueurs de se joindre à une course de kart en réalité mixte. Les joueurs ont la possibilité de récupérer des bonus ou des moyens d'entraver leurs adversaires. En suivant les mêmes principes que le jeu *Mario kart*, les bonus sont présents sur la piste et récupérables en les impactant. De-même, si un joueur entre en contact avec un obstacle virtuel placé par un autre joueur, son kart se voit alors ralenti pendant une courte durée par exemple.

6.4 Organisation

6.4.1 Organisation du projet

Ce projet est développé sur le centre de Sophia Antipolis, dans les Alpes Maritimes. L'équipe travaillant sur ce projet est composée en majeure partie de stagiaires élèves ingénieurs réalisant leur projet de fin d'études. Chacun possède une spécificité du projet à développer. Ils sont guidés et conseillés par des ingénieurs de l'entreprise, spécialisés dans les domaines étudiés. L'équipe KARlab que j'ai eu l'occasion d'intégrer est composée de 9 personnes : 1 Chef(fe) de projet, 2 ingénieurs à temps plein, 1 doctorant et 5 stagiaires, comme ci-dessous figure 8.

Initialement déployée en cycle en V, l'équipe KARlab évolue à présent en méthodologie AGILE SCRUM. Cette organisation est adaptée pour créer un produit à partir de demandes de clients. Pour cela, des personnes de l'équipe doivent jouer le rôle de *Product Owner* et de *SCRUM Master*. Le premier établit les spécificités du produit, les tâches à réaliser et valide les fonctionnalités développées. Le SCRUM master assure la communication entre les membres de l'équipe et prend en compte le retour sur expérience de chacun.

Le projet total est donc découpé en petites tâches appelées aussi *stories*. Toutes celles-ci sont réalisées en fonction de leur importance au cours des différents *sprints*, correspondant à des périodes de 3 semaines pour le

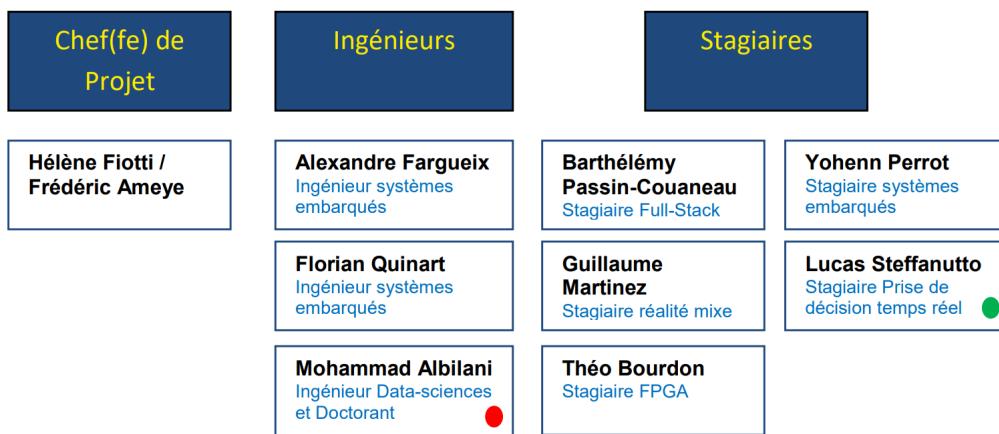


FIGURE 8 – Organisation du projet KARlab et encadrement du stage

projet KARlab.

À chaque début de sprint, un *sprint planning* est organisé : c'est une réunion permettant d'assigner un nombre de jours de travail associé à chaque tâche à réaliser, puis de les assigner aux membres de l'équipe pour le sprint. Au final, une *sprint review* est organisée pour voir quelles tâches ont été réalisées et leur avancement. Par la suite, une démonstration des fonctionnalités développées au cours du sprint est faite au client. Enfin, une réunion de *sprint retro* est organisée, de manière à avoir un retour sur expérience de chacun sur le sprint écoulé. Cette dernière permet de réfléchir sur les axes d'améliorations possibles dans le développement du produit. La méthode AGILE, est une méthode itérative et propose une livraison du produit en continu.

De plus, chaque matin, l'équipe se réunit pour un *Daily* de 15 minutes où chacun explique ses avancements de la veille et les difficultés rencontrées, ainsi que les travaux prévus du jour.

Pour finir, Les spécificités, études, développements et documentations sont stockées sur la plate-forme collaborative de l'entreprise, Confluence. L'outil de versionning utilisé est GitLab. Il permet aussi de gérer les sprints, les stories et les repositories.

6.4.2 Organisation de mon stage

Mon encadrement était assuré par mon tuteur technique Mohammad Albilani (pastille rouge Figure 8). Chaque lundi nous faisions un point technique sur l'avancement de la semaine passée et le travail prévu pour la semaine en cours. Certaines présentations plus détaillées du travail étaient aussi fixées de temps à autres. La Cheffe de projet et le responsable d'affaires associé étaient aussi en charge de la responsabilité de mon stage. Les missions qui m'étaient initialement confiées sont présentes sur le tableau 2 suivant :

Activités initialement confiées pour le stage
<ul style="list-style-type: none"> - Intégration dans le projet de l'entreprise - Compréhension du système et des enjeux du stage (embarqué, temps réel, algorithme final en C++) - Etat de l'art et montée et compétence sur les méthodes d'apprentissage automatique - Proposition et validation d'un modèle d'algorithme - Implémentation de l'algorithme d'apprentissage par renforcement (en python) - Tests et expérimentations avec plusieurs échantillons de données - Evaluations de performances, optimisation et validation - Recommandations pour la futur traduction en C++ - Rédaction de documentation (technique et démarche scientifique)

TABLE 1 – Prévision initial de mon stage sur le projet KARlab

7 Introduction à l'Apprentissage par Renforcement

L'Apprentissage par Renforcement (AR), aussi appelé Reinforcement Learning, est une des branches du Machine Learning. Il se distingue de l'apprentissage supervisé et de l'apprentissage non-supervisé : Il n'y a pas de supervision durant l'apprentissage, ni de base de données. L'algorithme se sert de ses interactions avec son environnement pour améliorer ses prises de décisions, et au fur et à mesure du temps, apprendre un comportement complexe.

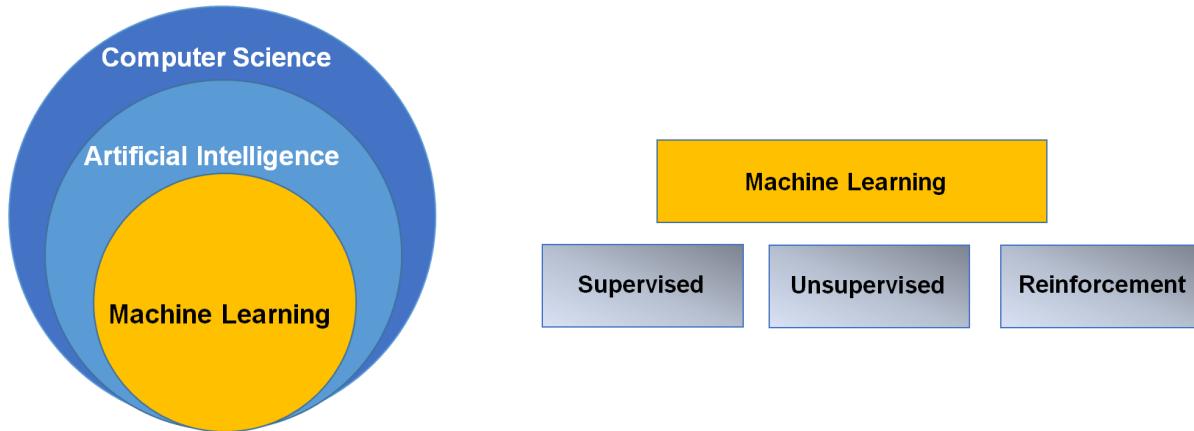


FIGURE 9 – Positionnement de l'AR par rapport au Machine Learning

7.1 Description

L'AR consiste à faire apprendre une tâche complexe à un agent autonome (robot, véhicule, kart KARlab, etc.). L'agent part donc d'un état initial jusqu'à atteindre un état terminal, qui peut être son but. Pour y parvenir l'agent va apprendre à réaliser certaines actions en fonction de l'état dans lequel il se trouve. Chaque instant où il réalise une action est marqué par une récompense, positive ou négative. L'agent observe les effets de ses actions, et améliorer ses décisions futures. L'apprentissage d'un comportement optimal va se faire grâce à des répétitions successives de la tâche : Son but est d'optimiser le gain final moyen qu'il peut espérer obtenir à la fin d'un épisode. La figure 10 à gauche illustre les interactions de l'agent avec son environnement.

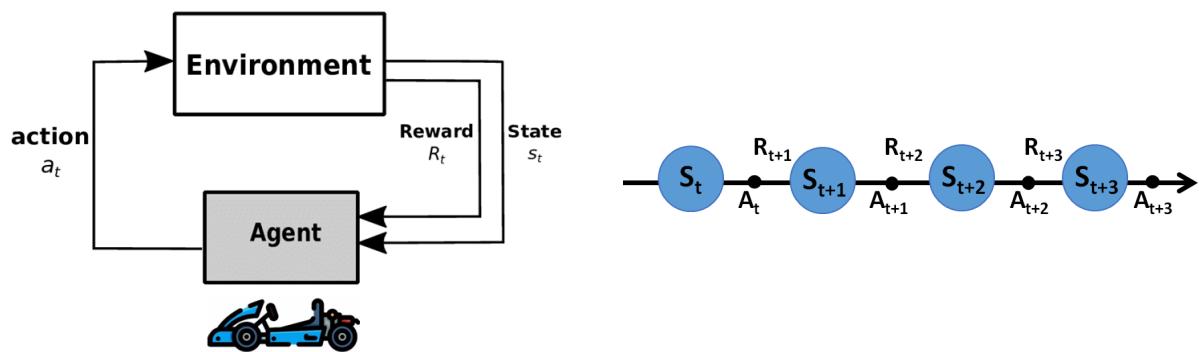


FIGURE 10 – Boucle d'interaction de l'agent avec son environnement (gauche) et schéma d'une trajectoire effectuée par un agent de l'instant t à $t+3$ (droite)

Sur la précédente, à gauche, l'agent se trouve dans un certain état S_t , effectue une action A_t depuis cette état, reçoit une récompense R_{t+1} puis passe dans un nouvel état S_{t+1} . Une trajectoire de l'agent est donc définie par une suite **État-Action-Récompense**. Ce sont ces séries qui constitueront les données d'apprentissage.

7.2 Modélisation

Ce problème d'optimisation est modélisé par un Processus de Markov Décisionnel (MDP). Il va nous permettre d'avoir un modèle formel capable de représenter la dynamique du monde dans lequel l'agent évolue, en émettant un jugement de valeur face aux situations dans lesquelles il pourrait se retrouver. Ses composantes principales sont les suivantes :

- **L'espace d'états S** du système, où chaque état s , est modélisé par un vecteur qui contient toutes les caractéristiques de l'environnement nécessaires et suffisantes pour prédire l'évolution future du système. Il peut comprendre des caractéristiques de l'agent : position, vitesse etc.
- **L'espace d'actions A** du système, est constitué des actions a que l'agent peut effectuer à partir d'un état donné. C'est la politique π d'un système qui va préciser à l'agent quelle action effectuer pour chaque état.
- **Une fonction de transition T : $S \times A \times S \mapsto [0, 1]$** qui donne pour chaque état, la probabilité T de passer d'un état actuel s , à l'état suivant s' , sachant que l'action a est effectuée.
- **Une fonction de récompense R : $S \times A \mapsto \mathbb{R}$** . Cette fonction permet de préciser à l'agent, combien sa décision prise à l'instant t est appréciée. La récompense peut être instantanée ou différée. Elle peut être représentée sous diverses formes, comme des points dans le cas d'un jeu vidéo. Pour l'exemple d'un drone : si le drone se crash, il obtient une récompense négative, si le drone suit la bonne trajectoire, il obtient une récompense positive.

Les notations S_t , A_t et R_t correspondent aux Variables Aléatoires prisent à l'instant t . Les notations s_t , a_t et r_t correspondent à leurs réalisations à l'instant t . De plus, la **propriété de Markov** précise qu'un état à l'instant t , ne dépend que de l'état précédent $t-1$. de cette manière on a : $P(S_t|S_0, S_1, \dots, S_{t-1}) = P(S_t|S_{t-1})$.

Remarque : Dans un problème d'apprentissage par renforcement réel, T et R sont à priori inconnues.

- **Un épisode** est l'ensemble des tuples états-actions-récompenses (s_t, a_t, r_t) obtenues à chaque instant, entre le point de départ de l'agent et son état terminal. Un épisode s'arrête lors de l'échec ou de la réussite de l'agent (ou même après avoir atteint une certaine limite de temps). T représente le temps pour atteindre un état terminal.
- **Le Gain, ou Retour**, est défini par le cumul des récompenses (ou "reward") immédiates perçues par le système, de l'instant présent, jusqu'à la fin de vie du processus [1] : $G = R_0 + R_1 + \dots + R_T$. Le retour est la mesure de performance associée à une séquence de transition donnée. Il est aussi possible d'introduire un facteur de dévaluation γ qui va pondérer les récompenses obtenues, de manière à leurs donner plus ou moins d'importance lorsqu'elles sont éloignées dans le temps. L'AR permet donc de s'adapter aux problèmes nécessitant un compromis entre récompense immédiate et récompense à long terme.
- **La politique π** est une fonction permettant d'associer une action à effectuer à partir d'un état donné. Il est noté : $\pi : S \mapsto A$. Elle définit le comportement que l'agent suit durant toute la durée de vie du processus. Une politique peut être déterministe ou stochastique, c'est à dire qu'elle peut associer une probabilité, à chaque action possible et réalisable depuis un état donné. On a alors : $\pi : S \times A \mapsto [0; 1]$

Le but de l'AR est d'obtenir une politique optimale, notée π^* qui définit l'action optimale à effectuer pour chaque état, de manière à, en moyenne, maximiser le retour final [2].

7.3 Résolution

7.3.1 Fonction de valeur d'état

La fonction de valeur d'état sous la politique π , est notée $V^\pi(s)$ (aussi $V(s)$ ou même V , par abus de langage). Elle correspond au retour moyen qu'il est possible d'espérer, en partant d'un état s , puis en suivant la

politique π jusqu'à la fin de l'épisode. Elle permet de mesurer la performance d'une politique. On a donc pour chaque état : $V : S \mapsto \mathbf{R}$, et donc, d'après les explications précédentes sur la définition du gain/retour) :

$$V^\pi(s) = E_\pi[R_t | s_t = s] = E_\pi\left[\sum_{n=1}^{\infty} \gamma^n r_{t+n+1} | s_t = s\right] \quad (1)$$

Cette formule est aussi décrite par l'équations de Bellman "Bellman Expectation" de la manière suivante :

$$V^\pi(s) = \sum_{a \in A(s)} \pi(s, a) [R_{sa} + \gamma \sum_{s' \in S^+} T_{sa}^{s'} V^\pi(s')] \quad (2)$$

Cette formule est basée sur des principes de Programmation Dynamique [2]. Elle consiste en une rétro-propagation de la valeur. Elle exprime la fonction de valeur de l'état actuel s à partir des fonctions de valeur des états suivants possibles s' , atteints en réalisant une action a , avec une probabilité de transition T . Les démonstrations peuvent être retrouvées dans le cours de [3], et peuvent aussi s'appliquer pour la fonction $Q(s, a)$.

7.3.2 Fonction de valeur d'état-action

La fonction de valeur d'état-action sous la politique π , est notée $Q^\pi(s, a)$ (aussi $Q(s, a)$ ou même Q , par abus de langage). Elle correspond au retour moyen qu'il est possible d'espérer, en partant d'un état s , en effectuant l'action a , puis en suivant la politique π jusqu'à la fin de l'épisode. On a : $Q : S \times A \mapsto \mathbf{R}$ et donc :

$$Q^\pi(s, a) = E_\pi[R_t | s_t = s, a_t = a] = E_\pi\left[\sum_{n=1}^{\infty} \gamma^n r_{t+n+1} | s_t = s, a_t = a\right] \quad (3)$$

7.3.3 Optimalité

Les deux fonctions précédentes peuvent être définies de manière optimale avec les formules suivantes :

$$V^*(s) = \max_{\pi} V(s) \quad (4)$$

$$Q^*(s, a) = \max_{\pi} Q(s, a) = E[r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a] \quad (5)$$

En appliquant l'équation (2), à celles (4) et (5), ces dernières deviennent les équations (6) et (7) suivantes :

$$V^*(s) = \max_{a \in A(s)} [R_{sa} + \gamma \sum_{s' \in S^+} T_{sa}^{s'} V^*(s')] \quad (6)$$

$$Q^*(s) = R_{sa} + \gamma \max_{a \in A(s)} \sum_{s' \in S^+} T_{sa}^{s'} Q^*(s', a') \quad (7)$$

Ces deux dernières équations peuvent être schématisées de manière plus digeste sur la figure suivante, où (a) et (b) correspondent respectivement à (6) et (7) :

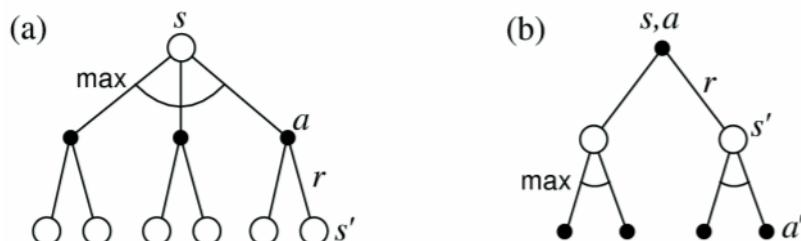


FIGURE 11 – Schéma des équations de Bellman optimales

7.4 Résolution et Programmation Dynamique

Une politique optimale est calculée, grâce au principe d'optimalité de Bellman. Ce dernier définit que toute politique optimale est construite à partir de sous-politiques optimales. La fonction $Q^*(s, a)$ ordonne l'ensemble des actions possibles pour chaque état accessible. Cette fonction va donc permettre d'obtenir π^* .

Note : Il peut exister plusieurs politiques optimales, mais elles ont toutes la même fonction de valeur d'état optimale $V^*(s)$ et toutes la même fonction de valeur état-action $Q^*(s, a)$.

L'extraction de π^* à partir de V^* se révèle plus compliquée puisque cela nécessite la fonction de transition T , qui n'est pas connue dans les problèmes de la vie réelle. Le principe de *model-free* est donc utilisé, se servant uniquement des pairs état-action-récompense au cours du temps, pour apprendre de son environnement. Néanmoins, il est nécessaire de comprendre d'abord le cas fondamental du principe *model-based*, où l'agent connaît tous les paramètres exacts du système. C'est le cas de la **Programmation Dynamique**.

La **Programmation Dynamique** est l'une des bases des principes d'apprentissage par renforcement. Elle peut être utilisée lorsque le modèle **MDP** est complètement connu (probabilités de transitions, récompenses, etc.). C'est donc plus un problème d'optimisation et de décision que d'apprentissage. Aussi appelé problème de planification, il peut être résolu en trouvant une politique optimale π^* que l'agent doit suivre. Le principe de la Programmation Dynamique consiste à diviser un problème en sous-problème à résoudre, en stockant les solutions.

Pour trouver une politique optimale, il faut d'abord évaluer combien une politique donnée est profitable.

7.4.1 Évaluation de la politique

Pour cela, il est possible de se baser sur la fonction de valeur, qui peut s'obtenir en adaptant l'équation de Bellman "*Bellman Expectation*" (2). Cette formule permet de faire remonter l'information de valeur des potentiels états suivants s' , au niveau de l'état actuel s , à partir de toutes les actions qu'il est possible de réaliser depuis cette état. Dans notre cas, le principe à réaliser est similaire, en ajoutant une mise à jour : La nouvelle valeur de l'état s , $V_{k+1}(s)$ est obtenue à partir de toutes les anciennes valeurs des états suivants possibles $V_k(s')$ et des récompenses immédiates associées. Cette formule est la suivante d'après [3] :

$$V_{k+1}(s) = E[r_{t+1} + \gamma V_k(s_{t+1})|s_t = s] = \sum_{a \in A(s)} \pi(s, a)[R_{sa} + \gamma \sum_{s' \in S^+} T_{sa}^{s'} V^\pi(s')] \quad (8)$$

Exemple concret : On considère un agent se déplaçant sur une grille de taille 4×4 , où chaque case constitue un état. L'agent doit se déplacer d'un état de départ, jusqu'à un état terminal (en jaune figure 12). Pour l'initialisation, une politique aléatoire est choisie (à chaque pas de temps, l'agent se déplace aléatoirement d'une case à une case voisine), ainsi qu'une fonction de récompense $R = -1$ et une fonction de valeur d'états initialisée à $V_0(s) = 0$ pour l'ensemble des états. Pour l'exemple, le calcul de la première mise à jour de $V_1(6)$ est développé sur la figure 12 suivante, d'après [2].

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

$$\begin{aligned}
 v_1(6) &= \sum_{a \in \{u, d, l, r\}} \pi(a|6) \sum_{s', r} p(s', r|6, a) [r + \gamma v_0(s')] \\
 &= \sum_{\substack{a \in \{u, d, l, r\} \\ 0.25 \forall a}} \pi(a|6) \sum_{s'} p(s'|6, a) [\underbrace{r}_{-1} + \gamma \underbrace{v_0(s')}_0] \\
 &= 0.25 * \{-p(2|6, u) - p(10|6, d) - p(5|6, l) - p(7|6, r)\} \\
 &= 0.25 * \{-1 - 1 - 1 - 1\} \\
 &= -1 \\
 \Rightarrow v_1(6) &= -1
 \end{aligned}$$

FIGURE 12 – Schéma et détail des calculs la mise à jour d'un état pour l'évaluation d'une politique d'un problème de Programmation Dynamique

Ce calcul est ensuite réalisé pour tous les états du système, afin d'obtenir la fonction de valeur $V_1(s)$. Il peut

être ensuite réitéré de nouveau pour tous les états, de manière à se rapprocher le plus possible de la fonction de valeur sous la politique actuelle $V^\pi(s)$, qui va nous permettre d'évaluer la politique actuelle π . La figure 13 suivante est un exemple de l'évolution de $V(s)$ au cours des itérations de l'exemple précédent :

$k = 0$	$k = 1$	$k = 2$																																																
<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr> <tr><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr> <tr><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr> <tr><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr> </table>	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td>0.0</td><td>-1.0</td><td>-1.0</td><td>-1.0</td></tr> <tr><td>-1.0</td><td>-1.0</td><td>-1.0</td><td>-1.0</td></tr> <tr><td>-1.0</td><td>-1.0</td><td>-1.0</td><td>-1.0</td></tr> <tr><td>-1.0</td><td>-1.0</td><td>-1.0</td><td>0.0</td></tr> </table>	0.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.0	<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td>0.0</td><td>-1.7</td><td>-2.0</td><td>-2.0</td></tr> <tr><td>-1.7</td><td>-2.0</td><td>-2.0</td><td>-2.0</td></tr> <tr><td>-2.0</td><td>-2.0</td><td>-2.0</td><td>-1.7</td></tr> <tr><td>-2.0</td><td>-2.0</td><td>-1.7</td><td>0.0</td></tr> </table>	0.0	-1.7	-2.0	-2.0	-1.7	-2.0	-2.0	-2.0	-2.0	-2.0	-2.0	-1.7	-2.0	-2.0	-1.7	0.0
0.0	0.0	0.0	0.0																																															
0.0	0.0	0.0	0.0																																															
0.0	0.0	0.0	0.0																																															
0.0	0.0	0.0	0.0																																															
0.0	-1.0	-1.0	-1.0																																															
-1.0	-1.0	-1.0	-1.0																																															
-1.0	-1.0	-1.0	-1.0																																															
-1.0	-1.0	-1.0	0.0																																															
0.0	-1.7	-2.0	-2.0																																															
-1.7	-2.0	-2.0	-2.0																																															
-2.0	-2.0	-2.0	-1.7																																															
-2.0	-2.0	-1.7	0.0																																															
$k = 3$	$k = 10$	$k = \infty$																																																
<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td>0.0</td><td>-2.4</td><td>-2.9</td><td>-3.0</td></tr> <tr><td>-2.4</td><td>-2.9</td><td>-3.0</td><td>-2.9</td></tr> <tr><td>-2.9</td><td>-3.0</td><td>-2.9</td><td>-2.4</td></tr> <tr><td>-3.0</td><td>-2.9</td><td>-2.4</td><td>0.0</td></tr> </table>	0.0	-2.4	-2.9	-3.0	-2.4	-2.9	-3.0	-2.9	-2.9	-3.0	-2.9	-2.4	-3.0	-2.9	-2.4	0.0	...	<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td>0.0</td><td>-6.1</td><td>-8.4</td><td>-9.0</td></tr> <tr><td>-6.1</td><td>-7.7</td><td>-8.4</td><td>-8.4</td></tr> <tr><td>-8.4</td><td>-8.4</td><td>-7.7</td><td>-6.1</td></tr> <tr><td>-9.0</td><td>-8.4</td><td>-6.1</td><td>0.0</td></tr> </table>	0.0	-6.1	-8.4	-9.0	-6.1	-7.7	-8.4	-8.4	-8.4	-8.4	-7.7	-6.1	-9.0	-8.4	-6.1	0.0																
0.0	-2.4	-2.9	-3.0																																															
-2.4	-2.9	-3.0	-2.9																																															
-2.9	-3.0	-2.9	-2.4																																															
-3.0	-2.9	-2.4	0.0																																															
0.0	-6.1	-8.4	-9.0																																															
-6.1	-7.7	-8.4	-8.4																																															
-8.4	-8.4	-7.7	-6.1																																															
-9.0	-8.4	-6.1	0.0																																															
	...	<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td>0.0</td><td>-14.</td><td>-20.</td><td>-22.</td></tr> <tr><td>-14.</td><td>-18.</td><td>-20.</td><td>-20.</td></tr> <tr><td>-20.</td><td>-20.</td><td>-18.</td><td>-14.</td></tr> <tr><td>-22.</td><td>-20.</td><td>-14.</td><td>0.0</td></tr> </table>	0.0	-14.	-20.	-22.	-14.	-18.	-20.	-20.	-20.	-20.	-18.	-14.	-22.	-20.	-14.	0.0																																
0.0	-14.	-20.	-22.																																															
-14.	-18.	-20.	-20.																																															
-20.	-20.	-18.	-14.																																															
-22.	-20.	-14.	0.0																																															
		$\leftarrow v_\pi$																																																

FIGURE 13 – Schéma de l'itération de l'évaluation d'une politique

Un critère de convergence peut être défini pour borner le nombre d'itérations k à effectuer.

7.4.2 Amélioration de la politique

Une fois que $V^\pi(s)$ est atteinte ou approchée, il faut vérifier si il est possible d'améliorer la politique associée. Le théorème d'amélioration de la politique, équation (9) suivante, doit donc être vérifié [3] :

$$\text{Si } \forall s \in S, Q^\pi(s, \pi'(s)) \geq V^\pi(s), \text{ alors } \forall s \in S, V^{\pi'}(s) \geq V^\pi(s), \text{ et } \pi' > \pi \quad (9)$$

Si c'est le cas, cela signifie qu'il existe une politique π' au moins aussi bonne que π , qui peut être obtenue de la manière suivante, à partir de la fonction $V^\pi(s)$ qui a été approché précédemment [3] :

$$\pi'(s) = \underset{a \in A(s)}{\operatorname{argmax}} Q^\pi(s, a) = \underset{a \in A(s)}{\operatorname{argmax}} E[r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s, a_t = a] = \underset{a \in A(s)}{\operatorname{argmax}} [R_{sa} + \gamma \sum_{s' \in S^+} T_{sa}^{s'} V^\pi(s')] \quad (10)$$

Une fois la politique améliorée, ce processus peut être réitéré depuis l'évaluation de politique de manière à approcher $V^{\pi'}(s)$, puis déterminer une potentielle nouvelle meilleure politique π'' . Au fur et à mesure de ces itérations, la politique que l'agent doit suivre va donc s'améliorer, et se rapprocher d'une politique optimale π^* .

En reprenant la première itération de ce processus pour l'exemple de la figure 12, et en supposant $V^\pi(s)$ approchée de manière optimale, d'après [2] l'équation précédente donne la politique π' représentée sur la grille de droite de la figure 14 de la page suivante.

Cette technique consiste donc en une boucle d'itérations successives “évaluation-amélioration”, répétée jusqu'à approcher au mieux une politique optimale π^* . L'inconvénient de cette méthode réside dans le fait qu'à chaque itération, une évaluation de politique doit être faite, et tous les états du système doivent être parcourus. Cela est répété plusieurs fois, jusqu'à approcher au mieux $V^\pi(s)$. Cela peut poser un problème de coût de calcul, dans le cas d'un système plus complexe que l'exemple abordé, si il contient un très grand nombre d'états et d'actions possibles. Une autre technique basée sur l'itération de valeur peut être une solution.

Amélioration de la politique par itération de valeur :

Il n'est pas nécessaire d'obtenir exactement $V^\pi(s)$, il est possible d'arrêter l'évaluation de politique plus tôt. Dans l'exemple de la figure 13, pour l'itération $k = 10$, il est déjà possible de trouver une politique optimale.

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

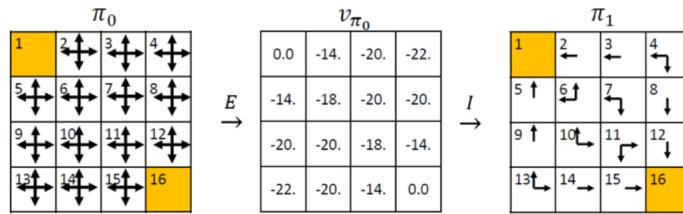


FIGURE 14 – Schéma de l'amélioration de politique de la première itération du processus, et chaîne des itérations jusqu'aux fonctions optimales

La méthode de l'amélioration de la politique par itération de valeur consiste donc à faire seulement une seule évaluation de politique, puis continuer le processus en itérant directement sur la fonction de valeur d'état obtenue. La même formule d'évaluation de politique est alors appliquée, à la différence quelle prend le maximum sur toutes les actions. Une fois qu'un critère de convergence est vérifié, alors la politique optimale π^* correspondante est obtenue en appliquant la même formule que vue précédemment permettant de calculer π' .

Ces approches “*model-based*” par Programmation Dynamique sont limitées. Elles nécessitent un coût calculatoire élevé, qui augmente avec le nombre d'états du système. Il est aussi nécessaire de connaître le système dans lequel l'agent évolue. Ce n'est pas le cas dans un problème d'AR réel, où la fonction de transition T , et la fonction de récompense R sont inconnues. Néanmoins, ce manque d'information peut être compensé par l'interaction de l'agent avec son environnement.

8 Algorithmes classiques

De façon analogue à la Programmation Dynamique, il existe deux approches principales permettant de trouver une politique optimale π^* pour résoudre un problème d'apprentissage par renforcement. L'une est basée sur l'itération successive de la politique π , afin de converger directement vers π^* . La seconde réalise des itérations sur la fonction de valeur $Q(s, a)$, pour converger vers son optimum Q^* . Cette famille de méthode est aussi connue sous le nom de Q-Learning. Dans les deux cas, le concept de “*model-free*” énoncé s'applique.

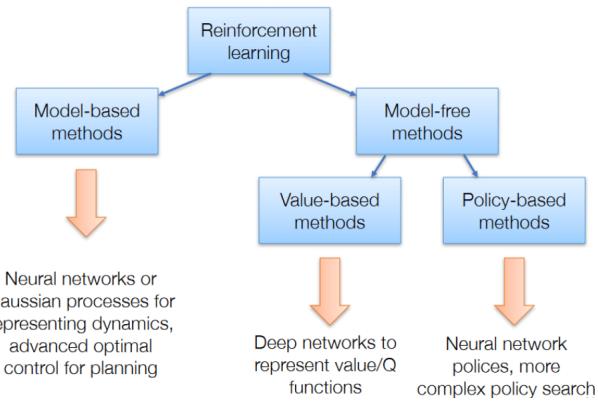


FIGURE 15 – Schéma de l'organisation des approches d'Apprentissage par Renforcement

8.1 Approches basées politique

Deux approches de cette catégorie seront décrites ici, une première assez classique, puis une seconde plus élaborée.

8.1.1 Monte Carlo Learning

Les méthodes de Monte-Carlo recueillent toutes les informations d'un épisode en stockant les épisodes complets : ce sont ces données sur lesquels l'agent va réaliser l'apprentissage. Il est important de noter que dans ce cas, il est obligatoire d'attendre la fin de l'épisode pour pouvoir apprendre à partir du gain final (cf. "Monte Carlo Learning" figure 16). L'agent ne peut donc pas apprendre directement après chaque action effectuée.

Le système est initialisé avec une politique π et une fonction de valeur d'état V quelconques. La méthode de Monte Carlo fonctionne en deux étapes similaires à celles de la Programmation Dynamique : on évalue d'abord la politique courante, puis on l'améliore.

Evaluation de Monte Carlo de la politique : La fonction de valeur $V^\pi(s)$ est estimée en faisant une moyenne des retours obtenus, en générant un très grand nombre d'épisodes. Pour rappel, le retour R , correspond à la somme des récompenses immédiates obtenues durant un épisode. Dans l'apprentissage de Monte Carlo, il y a deux façons d'effectuer le moyennage permettant d'estimer $V^\pi(s)$, et donc d'évaluer la politique actuelle π [4]. Au final, $V(s)$ est mise à jour après chaque épisode de la manière suivante :

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)] \quad (11)$$

Amélioration de Monte Carlo de la politique : Dans le cas de la **Programmation Dynamique**, l'équation (2) "Bellman Expectation" pouvait être utilisée. Ici ce n'est pas possible car la fonction de transition T n'est pas disponible. Néanmoins, la même formule sera appliquée, en choisissant d'appliquer une politique déterministe prenant à chaque fois l'action optimale pour chaque état. On appelle "greedy" ce genre de politique.

$$\pi(s) = \operatorname{argmax}_{a \in A(s)} Q^\pi(s, a) \quad (12)$$

Le théorème d'amélioration de politique peut donc s'appliquer aussi dans ce cas, en injectant l'équation précédente pour choisir les actions de $Q(s, a)$ sous la politique π_k de l'équation suivante :

$$\begin{aligned} Q^{\pi_k}(s, \pi_{k+1}(s)) &= Q^{\pi_k}\left(s, \arg \max_{a \in \mathcal{A}(s)} Q^{\pi_k}(s, a)\right) \\ &= \max_{a \in \mathcal{A}(s)} Q^{\pi_k}(s, a) \\ &\geq Q^{\pi_k}(s, \pi_k(s)) \\ &= V^{\pi_k}(s) \end{aligned} \quad (13)$$

Cette manière de procédé nécessite l'hypothèse d'avoir des actions exploratoires en début d'épisodes [3]. Il est donc intéressant de s'affranchir de cette nécessité, en introduisant une part d'aléatoire dans le choix des actions : Ce genre de politique est appelée "epsilon-greedy".

Les avantages du *Monte Carlo Learning* sont de ne pas avoir de biais, de bonnes propriétés de convergence, et de ne pas être sensible aux valeurs d'initialisation. Néanmoins, l'apprentissage de l'agent ne peut se faire qu'à chaque fin d'épisode pour la séquence complète, et donc ne fonctionne que pour des environnements épisodiques (la tâche est réussie ou non). Cette méthode présente aussi une variance élevée, qui peut allonger le temps de convergence.

8.1.2 Monte Carlo Policy Gradient

Cette méthode approxime la politique et la fonction de valeur, de manière à réduire le coût calculatoire, et réduire le *problème de la dimension* : Le nombre d'états (et d'actions) du système, peut vite devenir très grands dans certains problèmes.

Ici, π^* est directement recherchée dans l'espace des politiques, en moyennant les réalisations. La politique est paramétrisée par un réseau de neurones profond (cf. figure 41 annexe), pour faire face au problème de la dimension : On peut l'écrire π_θ ou $\pi(a|s; \theta)$, où θ représente un vecteur contenant les paramètres d'une

fonction d'approximation ou les poids d'un réseau de neurones. La fonction de perte suivante est alors définie :

$$J(\theta) = E_{\pi_\theta}[R] = E_{\pi_\theta}[R_1 + \gamma R_2 + \gamma^2 R_3 + \dots | \pi(\cdot; \theta)] \quad (14)$$

Elle représente le cumul des récompenses décomptées en suivant la politique courante paramétrisée $\pi(\cdot; \theta)$. Cette fonction de perte doit être optimisée, par *ascension de gradient stochastique* (méthode analogue au *Stochastic Gradient Descent* (SGD), figure 43 annexe). Cela va permettre l'ajustement du vecteur de poids θ . Cette maximisation est réalisée par le biais du *Théorème du Gradient de la politique* ci-dessous [5] :

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)] \quad (15)$$

Le gradient du *log* de la politique traduit à quel point une action est désirable, et la fonction Q sous la politique courante traduit l'intérêt de réaliser une action. Ici, l'idée est de dériver la fonction de perte J dans la direction de chacune des composantes de θ .

Dans l'équation ci-dessus, Q est approximée grâce à l'estimation R de la fonction V sous la politique courante π_θ , grâce au cumul des récompenses obtenues à partir de n'importe quel état.

De cette manière on pourra incrémenter θ , avec la formule suivante, où α représente une constante d'apprentissage aussi appelée "step-size" :

$$\Delta\theta = \alpha \nabla_\theta J(\theta) \quad (16)$$

Cet algorithme, connu aussi sous le nom de **REINFORCE** fonctionne donc de la manière suivante [4] :

- **On initialise de manière aléatoire** le vecteur θ de la politique.
- On génère un **épisode complet avec la politique courante** π_θ
- Pour **chaque pas de temps** de l'épisode : On estime V avec le retour moyen attendu R On **mets à jour** le paramètre de la politique : $\theta \leftarrow \theta + \Delta\theta$
- On répète le point 2 en **regénérant des épisodes** jusqu'à ce que π_θ approche π^* de manière convenable.

Les inconvénients de cette méthodes sont que l'agent doit attendre la fin d'un épisode avant de pouvoir apprendre et mettre à jour le paramètre θ . De plus, la variance du gradient est élevée, ce qui peut prendre d'autant plus de temps à converger. Il est aussi possible de rester bloqué dans un maximum local.

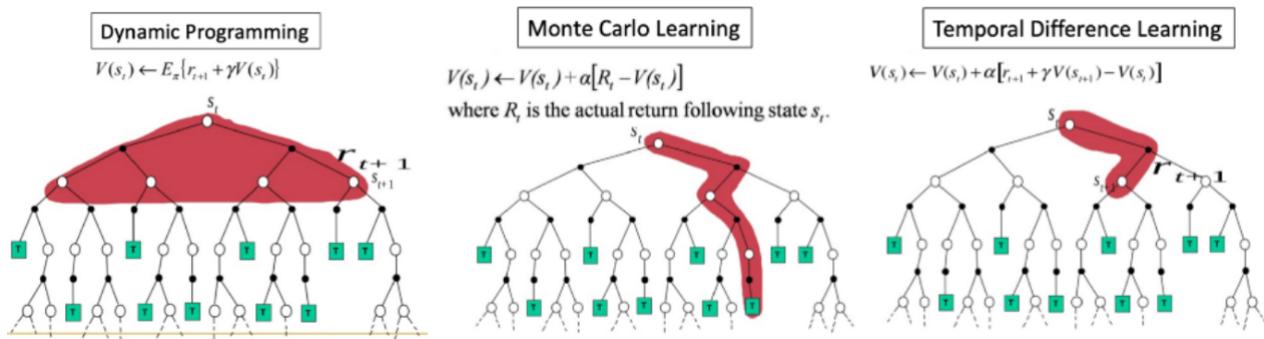


FIGURE 16 – Schéma de l'organisation des approches au sein du Renforcement Learning

Le Figure 16 ci-dessus est un exemple de schéma résumant les différentes approches expliquées jusqu'à présent. Les points blancs représentent les états dans lesquels l'agent se trouve, les points noirs les actions choisies. Un carré vert représente un état terminal. Le rouge représente le chemin suivi par l'agent, avant de mettre à jour sa politique, par la méthode d'actualisation écrite au-dessus de chaque arbre.

8.2 Approches basées valeur

Deux approches de cette catégorie seront décrites ici : la première plutôt classique, puis la seconde plus développée. Les mêmes hypothèses de “*model-free*” sont admises que précédemment.

8.2.1 Temporal Difference Learning (TD-Learning)

Ici, à l'inverse du Monte Carlo Learning, où l'agent doit attendre la fin d'un épisode avant d'apprendre, il peut mettre à jour directement sa fonction de valeur d'état $V(s)$ après chaque action réalisée 16. L'agent va se baser sur une différence de valeur entre deux états. Grâce à cette technique il ne sera pas nécessaire de stocker un épisode complet. L'agent incrémentera son apprentissage au fur à mesure des déplacements à travers les différents états du système, grâce à la mise à jour suivante :

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)] \quad (17)$$

Dans cette formule, le terme de droite entre crochets est aussi appelé **Erreur TD (TD Error)**.

8.2.2 Q-Learning

La méthode du Q-Learning est basée sur le principe du “*off-policy*” learning, où l'agent apprend une politique π en se basant sur une autre politique μ . Un exemple du principe de “*off-policy*” peut être imaginé en pensant à une personne qui apprend de l'expérience d'une autre, et donc sans réitérer les mêmes erreurs. C'est-à-dire que la politique utilisée pour agir et choisir une action est séparée de la politique évaluée [3].

Dans cette approche la politique optimale est obtenue par une approximation de la fonction de valeur-état optimale Q^* , afin de retrouver π^* . Cette méthode repose sur les équations de Bellman et le principe de Temporal Différence (TD) : Elle consiste à mettre à jour la fonction de valeur état-action optimale d'un état s sachant qu'on exécute l'action optimale a' lorsqu'on se trouve dans l'état suivant s' . Comme précédemment, la fonction de valeur Q peut être mise à jour après chaque action :

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a' \in A(s')} Q(s', a') - Q(s, a)] \quad (18)$$

Néanmoins, les méthodes de Q-Learning classiques stockent la fonction $Q(s, a)$ sous forme tabulaire, figure 17, en haut à droite). Elle est donc exposée au problème de la dimension, dans le cas d'un nombre d'états important.

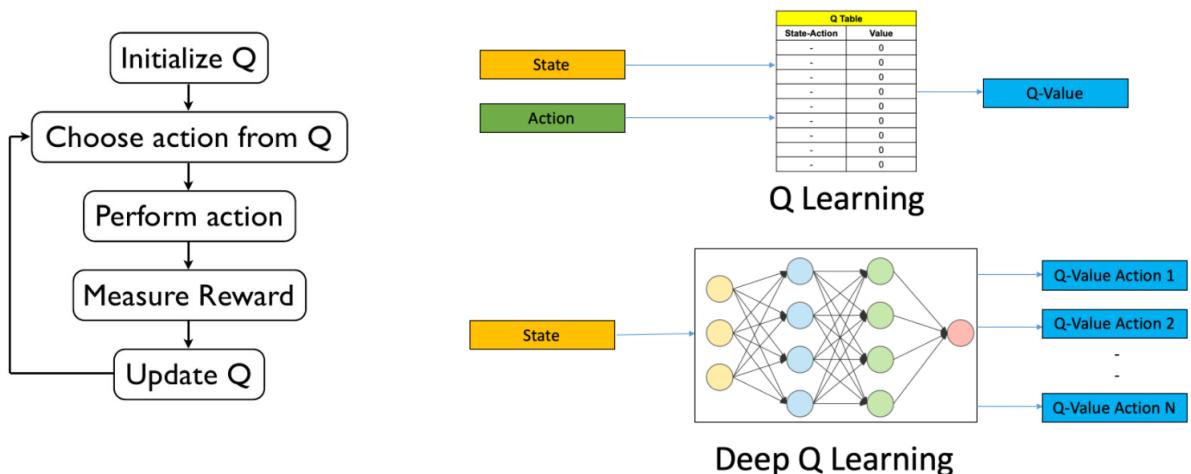


FIGURE 17 – Schéma du fonctionnement du Q-Learning (gauche) et différences avec le Deep Q-Learning (droite)

Note : Pour garantir l'exploration de l'algorithme, une politique “Epsilon-Greedy” est utilisée au niveau du terme qui prend le maximum de Q sur les actions.

8.2.3 Deep Q-Learning

Dans cette méthode, une représentation de la fonction de valeur-état $\mathbf{Q}(s, a; \mathbf{w})$ est construite, grâce à une paramétrisation de \mathbf{Q} , avec un **réseau de neurone profond (DQN)**, dont les paramètres sont contenus dans un vecteur \mathbf{w} . Dans le cadre d'actions discrètes, le réseau prend en entrée un état, et produit en sortie une estimation de la fonction de valeur état-action, pour toutes les paires possibles. Il est donc possible de sélectionner la meilleure action possible à effectuer à partir d'un état, à l'issue de cette estimation de \mathbf{Q}^* , figure 17, bas. L'*erreur quadratique moyenne* (MSE) est utilisée comme fonction de coût :

$$J(w) = [Q(s, a; \mathbf{w}) - (r + \gamma \max_{a' \in A(s')} Q(s', a'))]^2 \quad (19)$$

La formule (19) ci-dessus représente l'écart quadratique, entre l'estimation réel de la fonction \mathbf{Q} paramétrisée, et la fonction \mathbf{Q}^* que l'on doit approcher. Comme précédemment, le but est de minimiser cette écart par descente de gradient stochastique (SGD)

Dans cette méthode, le principe d'“*experience replay*” est aussi introduit, de manière à limiter la corrélation entre les échantillons durant l'entraînement : Certaines séquences (état,action,état,récompense) déjà explorées, sont repassées de manière aléatoire en entrée du réseau. De plus, pour limiter la non-stationnarité durant l'apprentissage, et optimiser au mieux la fonction de coût, le vecteur de poids cible \mathbf{w} est gelé au sein de la fonction $\mathbf{Q}(s', a'; \mathbf{w}^-)$. De cette manière la fonction de valeur peut être optimisée par rapport à une cible fixe.

Comme vu dans l'introduction, une fois que la fonction \mathbf{Q}^* est approchée, la politique optimale est obtenue de la manière suivante :

$$\pi^*(s) = \max_{a' \in A(s)} Q^*(s, a) \quad \forall s \in S \quad (20)$$

Résumé :

- **Dynamique Programming** : Correspond au cas “*model-based*”, dans un cas idéal, où tous les paramètres du MDP sont connus (probabilités de transition, fonction de récompense etc.). Des algorithmes de Programmation Dynamique de type évaluation-amélioration, fondés sur les équations de Bellman sont utilisés dans ce cas, pour approcher π^* .
- **Monte Carlo Learning** : représente le cas de l'**approche basée politique**, où l'on moyenne des réalisations d'épisodes pour approcher la politique optimale π^* .
- **Temporal Difference Learning** représente le cas de l'**approche basée valeur**, où la méthode est étendue au Q-Learning et au DQN, pour approcher π^* .

8.3 Approches mixtes

8.3.1 Actor-Critic

Ce principe permet de combiner les meilleures parties des deux approches abordées précédemment, l'une basée sur la valeur (Q-Learning 8.2.2), et l'autre basée sur le gradient de politique. L'algorithme approxime à la fois une fonction de valeur \mathbf{V} ou \mathbf{Q} avec un paramètre \mathbf{w} , ainsi que la politique π avec un paramètre θ , qui doit être maximisé. Ces méthodes basées sur le TD-Learning (8.2.1), représentent la politique indépendamment de la fonction de valeur [6].

Cet algorithme repose sur une approximation du Théorème du Gradient de la politique abordé précédemment, équation (15). Dans la fonction de perte, \mathbf{Q} est approximé par \mathbf{Q}_w comme suit [5] :

$$\begin{aligned} \nabla_\theta J(\theta) &\approx E_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)] \\ \Delta\theta &= \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a) \end{aligned} \quad (21)$$

De la même manière, une ascension de gradient doit être réaliser afin de maximiser le retour moyen de la politique π_θ , pour pouvoir se rapprocher au plus de θ^* .

L'algorithme fonctionne de la manière suivante :

- s , θ et w sont initialisés de manière aléatoire.
- Une première action a est effectuée selon la politique π_θ .
- Un épisode est réalisé par l'agent, et à chaque pas de temps $t = 1, \dots, T$:
 - 1. L'agent reçoit la récompense associée r ainsi que le nouvel état s' .
 - 2. L'action suivante à effectuer a' depuis ce nouvel état s' est choisie en suivant la politique π_θ
 - 3. Le paramètre θ de la politique est mis à jour : $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \ln \pi_\theta(a|s)$
 - 4. La TD error de la fonction de valeur utilisée dans l'algorithme est calculée : $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$
 - 5. La paramètre w de la fonction de valeur est mis à jour : $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$
- On recommence le processus en actualisant : $a \leftarrow a'$ $s \leftarrow s'$

Dans cette algorithme, α_w et α_θ sont des constantes d'apprentissage à prédéfinir pour la politique.

Au sein de la boucle, les points **2** et **3** sont réalisés par l'*Actor* tandis que les points **4** et **5** sont réalisées par le *Critic*. L'Actor prend les décisions à effectuer en se basant sur la politique π_θ , puis la mets à jour par la méthode du gradient de la politique. Le Critic, va évaluer l'action de l'Actor grâce à la TD error, puis mets à jour la fonction de valeur paramétrisée par w . De cette manière, à chaque pas de temps, l'agent peut réaliser un apprentissage de la fonction de valeur et de la politique, en incrémentant Q_w et π_θ . Le procédé est illustré à gauche, figure 18 suivante.

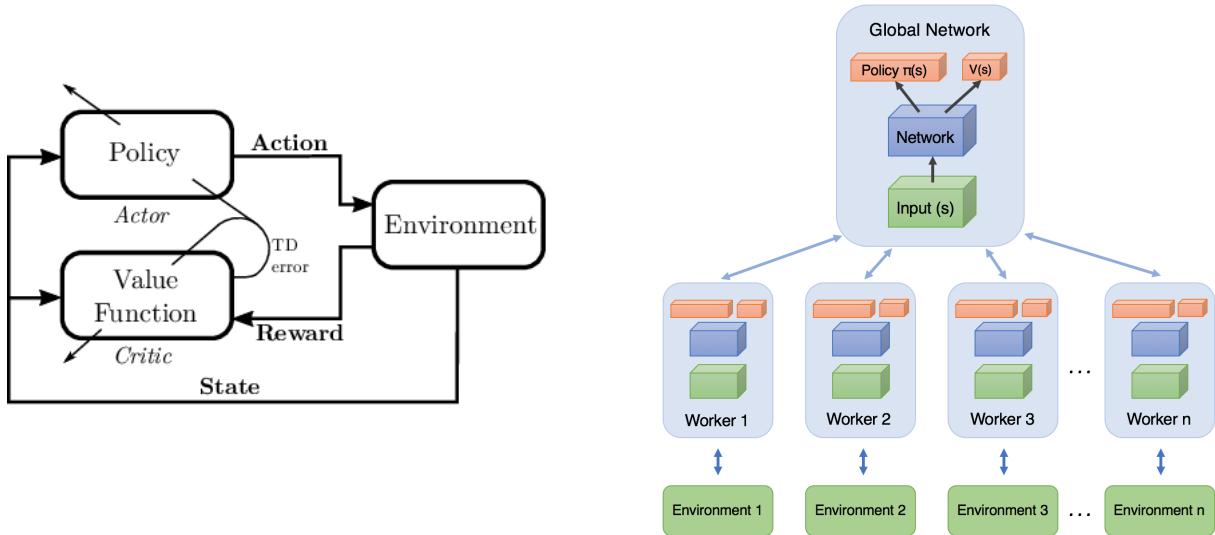


FIGURE 18 – Schéma du fonctionnement de l'apprentissage d'un agent avec la méthode Actor-Critic (gauche), et schéma de l'architecture de l'algorithme A3C (droite)

Comme pour les approches précédentes, il est possible de réduire le coût calculatoire, en réalisant des approximations non-linéaires des fonctions de valeur et de la politique : Les paramètres w et θ , peuvent être donc aussi des vecteurs comportant les poids d'un réseau de neurones. La méthode qui suit utilise cette technique.

8.3.2 Asynchronous Advantage Actor-Critic (A3C)

L'algorithme A3C, développé par l'équipe Deep Mind de Google [7], se base sur le travail de plusieurs agents en parallèle. Il peut être décrit comme l'utilisation parallélisée de la méthode Actor-Critic, à l'exception qu'ici, c'est la fonction de valeur V qui est approximée.

Ce modèle est composé d'un réseau global et de n agents, "workers" figure 18, où chacun dispose d'une copie de l'environnement, avec laquelle il va interagir de son côté par la méthode Actor-Critic. L'apprentissage des agents étant indépendant, cela permet de couvrir une plus grande partie de l'espace d'état S , et donc d'accélérer

la phase d'apprentissage. Le réseau global, qui contient les paramètres partagés entre les agents, est mis à jour de manière asynchrone : Les agents incrémentent le réseau global après un pas temps t_{max} propre à chacun d'entre eux. Chaque agent réinitialise ensuite ses paramètres avec ceux du réseau global avant de poursuivre l'apprentissage de son côté. Chaque fois qu'un agent apprend, il incrémente le réseau global et permet donc à tous les agents d'apprendre aussi : comme les humains chacun apprend de ses propres expériences, mais aussi de celles de ses pairs.

Comme précédemment, la politique optimale est approximée en paramétrisant la politique du réseau global π_θ avec un vecteur θ qui contient les poids d'un réseau de neurones profond. La fonction de valeur d'état du réseau global V_w est paramétrisée avec le même réseau d'apprentissage profond mais avec un paramètre w différent. Durant l'apprentissage, ils seront mis à jour l'un après l'autre, à chaque pas de temps. De la même manière, les paramètres locaux θ' et w' de chaque agent sont définis travaillant en parallèle dans leur propre copie de l'environnement.

De plus, dans cet algorithme, la notion d'*Advantage* est introduite. Le Critic va se servir de cette fonction pour évaluer l'action effectuée par l'Actor. Généralement, dans la mise en œuvre du Gradient de Politique, les récompenses permettent de montrer à l'agent lesquelles de ses actions ont été récompensées et lesquelles ont été pénalisées. Dans l'algorithme A3C, la fonction d'Advantage retourne une valeur qui permet de montrer à l'agent, combien une action réalisée est meilleure par rapport aux autres actions possibles, par rapport à la valeur moyenne attendue de cet état. Cela permet à l'agent d'avoir un nouvel aperçu de l'environnement, et donc d'améliorer le processus d'apprentissage. La mesure de l'Avantage est donnée par l'expression suivante :

$$A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s) \quad (22)$$

D'après [7], la fonction V de cette formule peut être vue comme une "*baseline*", une moyenne de la récompense perçue à l'état s associé. De plus, Q est remplacé par R , son estimateur non-biaisé. L'utilisation de l'Advantage réduit la variance du gradient de la politique stochastique.

Dans le papier originel de la méthode A3C, les auteurs ajoutent aussi un terme de régularisation H à la fonction de coût, qui a pour but de maximiser l'entropie de la politique pour améliorer l'exploration et décourager toute convergence prématurée [7]. Le gradient final de la fonction de coût s'exprime alors de la façon suivante :

$$\nabla J(\theta') = \nabla_{\theta'} \log \pi_{\theta'}(s, a) (R_t - V_w(s)) + \beta \nabla_{\theta'} H(\pi_{\theta'}(s, a)) \quad (23)$$

Où θ' représente le vecteur de poids de la politique locale d'un des n agents travaillant en parallèle, H l'entropie et w le vecteur de poids de la fonction de valeur du réseau global.

L'algorithme final au sein de chaque agent incrémentant le réseau global , "Worker 1 à Worker n" de la figure 18, est donc le suivant, figure 19 :

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

```

// Assume global shared parameter vectors  $\theta$  and  $\theta_v$  and global shared counter  $T = 0$ 
// Assume thread-specific parameter vectors  $\theta'$  and  $\theta'_v$ 
Initialize thread step counter  $t \leftarrow 1$ 
repeat
    Reset gradients:  $d\theta \leftarrow 0$  and  $d\theta_v \leftarrow 0$ .
    Synchronize thread-specific parameters  $\theta' = \theta$  and  $\theta'_v = \theta_v$ 
     $t_{start} = t$ 
    Get state  $s_t$ 
repeat
    Perform  $a_t$  according to policy  $\pi(a_t | s_t; \theta')$ 
    Receive reward  $r_t$  and new state  $s_{t+1}$ 
     $t \leftarrow t + 1$ 
     $T \leftarrow T + 1$ 
until terminal  $s_t$  or  $t - t_{start} == t_{max}$ 
 $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t; \theta'_v) & \text{for non-terminal } s_t // \text{Bootstrap from last state} \end{cases}$ 
for  $i \in \{t - 1, \dots, t_{start}\}$  do
     $R \leftarrow r_i + \gamma R$ 
    Accumulate gradients wrt  $\theta'$ :  $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i | s_i; \theta')(R - V(s_i; \theta'_v))$ 
    Accumulate gradients wrt  $\theta'_v$ :  $d\theta_v \leftarrow d\theta_v + \partial(R - V(s_i; \theta'_v))^2 / \partial \theta'_v$ 
end for
Perform asynchronous update of  $\theta$  using  $d\theta$  and of  $\theta_v$  using  $d\theta_v$ .
until  $T > T_{max}$ 

```

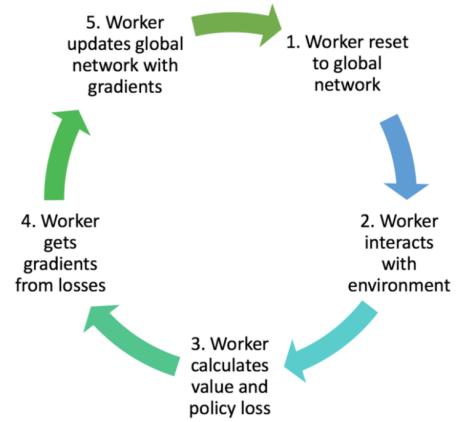


FIGURE 19 – Pseudocode de l'algorithme A3C, au sein de chacun des agents travaillant en parallèle (gauche) et schéma du fonctionnement global de l'algorithme A3C (droite)

Dans l'exemple du papier [7], les paramètres θ et w partagent le même réseau de neurone, mais ont chacun une sortie différente, respectivement *softmax* et *linéaire*.

La plupart des algorithmes présentés jusqu'à présent ne sont testés et évalués que sur des benchmarks classiques, et il y a peu ou pas d'applications réelles. L'état de l'art du stationnement autonome par apprentissage par renforcement, abordé par la suite section 10, montre que d'autres algorithmes sont utilisés pour la résolution de problèmes réels plus complexes, avec une application réelle. Dans la partie suivante, seul l'algorithme choisit au final pour son utilisation dans la problématique de KARlab sera décrit en détails. Les autres sont seulement décrits brièvement.

9 Algorithmes État de l'Art

Sur cette page seront décrits des algorithmes d'AR plus récents : Le Proximal Policy Optimization (PPO) et le Deep Deterministic Policy Gradient (DDPG), sont notamment utilisés dans des problématiques de conduite autonome.

9.1 Proximal Policy Optimization (PPO)

Dans le cas où une approche basée politique ou mixte est utilisée, cet algorithme permet d'obtenir un apprentissage plus robuste, stable et rapide. Le papier officiel [8], explique que le principe consiste à alterner interactions avec l'environnement, et optimisation d'une fonction de coût objective de "substitution" (*surrogate objective function*), en utilisant l'ascension de gradient stochastique. Les auteurs proposent une nouvelle fonction de coût permettant de multiples *Epochs* de mises à jour par *Mini-Batchs*. (au lieu d'une mise à jour de gradient par échantillonnage de données).

Note : En Machine Learning, un Epoch est un hyperparamètre. Il est défini comme le nombre de fois où tous les échantillons, soit la base de donnée entière, est passée à l'algorithme pour la mise à jour de ses paramètres internes (i.e. son apprentissage). Un Batch est un aussi un hyperparamètre. Il représente le nombre d'échantillons passés à l'algorithme avant la mise à jour de ses paramètres internes. Un Mini-Batch est donc un sous-ensemble de données d'un Epoch, d'après [9].

L'algorithme PPO est basée sur le Trust Region Policy Optimization [10], mais propose une implémentation simplifiée grâce à une optimisation du premier ordre. Dans l'algorithme TRPO, l'idée est d'augmenter la stabilité de la politique lors de l'apprentissage, en se basant sur une "région de confiance", et en évitant l'actualisation trop brusque des paramètres, qui pourrait changer la politique de manière trop importante en une seule fois. Dans la méthode TRPO, une contrainte de Divergence de Kullback-Liebler (DKL) est donc appliquée sur la taille de la mise à jour de la politique, à chaque itération.

La DKL [11], ou encore entropie relative, permet de mesurer la dissimilarité entre deux distributions de probabilités, soit deux politiques dans le cas présent. Le TRPO maximise une fonction de coût de substitution sous une contrainte de la taille de l'actualisation de la politique :

$$\begin{aligned} \text{maximize}_{\theta} & \quad \hat{E}_t \left[\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} \hat{A}_t \right] \\ \text{subject to} & \quad \hat{E}_t [\text{KL} [\pi_{\theta_{\text{old}}} (\cdot | s_t), \pi_{\theta} (\cdot | s_t)]] \leq \delta \end{aligned} \quad (24)$$

où θ_{old} est le vecteur de paramètres de la politique avant son actualisation.

Dans le papier de TRPO, il est au final suggéré d'utiliser une pénalité, plutôt qu'une contrainte, pour résoudre seulement un problème d'optimisation. Dans la formule (24) précédente, les auteurs soustraient au final le terme de contrainte avec un facteur β , à la première équation à maximiser.

L'algorithme PPO propose d'utiliser plutôt le ratio entre la politique actuelle et son actualisation. Il est noté :

$$r_t(\theta) = \frac{\pi_{\theta}(s_t, a_t)}{\pi_{\theta_{\text{old}}}(s_t, a_t)} \quad (25)$$

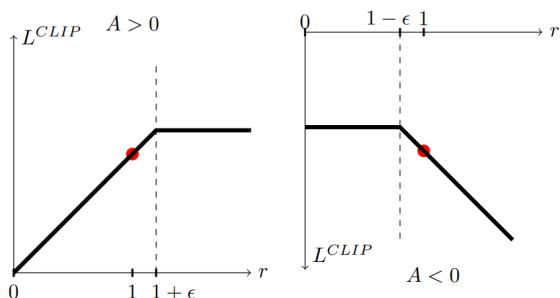
L'innovation de la méthode PPO est alors d'introduire une *fonction de coût de substitution écrêtée* ("Clipped Surrogate Objective") [8] :

$$L^{CLIP}(\theta) = \hat{E}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip} (r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right] \quad (26)$$

Où $\epsilon = 0, 2$ est un hyperparamètre. Cette formule peut être décrite de la façon suivante :

- Le premier terme dans la fonction *min* contient le ratio entre les politiques, et l'estimation de la fonction d'avantage décrite précédemment.
- Le second terme dans la fonction *min* contient aussi la fonction d'avantage, ainsi que la fonction "*clip*" permettant d'écrêter le ratio des probabilités. Ainsi, la possibilité que $r_t(\theta)$ se retrouve hors de l'intervalle $[1 - \epsilon; 1 + \epsilon]$ est supprimée.
- La fonction de coût finale est l'espérance du minimum de ces deux fonctions de coût.

Cette fonction de coût permet de **forcer le ratio entre la politique actuelle et la nouvelle politique, à être proche de 1** (précisément dans l'intervalle $[1 - \epsilon; 1 + \epsilon]$). La figure du papier [8] illustrant la fonction $L^{CLIP}(\theta)$ est la suivante, à gauche, figure 20 :



Algorithm 1 PPO, Actor-Critic Style

```

for iteration=1,2,... do
  for actor=1,2,...,N do
    Run policy  $\pi_{\theta_{\text{old}}}$  in environment for  $T$  timesteps
    Compute advantage estimates  $\hat{A}_1, \dots, \hat{A}_T$ 
  end for
  Optimize surrogate  $L$  wrt  $\theta$ , with  $K$  epochs and minibatch size  $M \leq NT$ 
   $\theta_{\text{old}} \leftarrow \theta$ 
end for

```

FIGURE 20 – Graphique expliquant la fonction $L^{CLIP}(\theta)$ (gauche) et pseudocode de l'algorithme PPO (droite)

Figure 20 à gauche ci-dessus, le point rouge montre le point de départ de la politique avant optimisation. Le graphique montre un seul pas de temps de la fonction de substitution L^{CLIP} en fonction du rapport r des lois de probabilités de chaque politique, pour un *Avantage* positif et un *Avantage* négatif.

Dans le cas de l'utilisation d'une architecture de réseau de neurones qui partage ses paramètres entre la politique (Actor) et la fonction de valeur (Critic) (cas *), [8] précise qu'en addition à la fonction de récompense, il est préférable d'ajouter un terme d'erreur (Erreur Quadratique Moyenne (MSE)) sur l'estimation de la fonction de valeur (second terme $L_t(\theta)$ de l'équation (27) suivante). Pour améliorer cette formule, un terme d'entropie H (similaire à celui du A3C) est aussi ajouté pour encourager l'exploration. Les coefficients c_1 et c_2 sont à ajuster dans le cas (*). La fonction de coût final qui s'inspire de l'algorithme A3C [7] décrit précédemment s'exprime comme ceci :

$$L_t^{CLIP_{final}}(\theta) = \hat{E}_t [L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 H[\pi_\theta](s_t)] \quad (27)$$

Les auteurs se basent aussi sur l'estimation de la fonction d'*Avantage* de la méthode A3C, qui consiste à approximer A avec les échantillons collectés sur T pas de temps . Cet estimateur est décrit de la manière suivante :

$$\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \dots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T) \quad (28)$$

Où t représente un index dans $[0, T]$, la longueur du segment de trajectoire.

Ils proposent alors d'utiliser la version tronquée suivante, qui peut être vue comme un mélange entre l'estimation précédente (28), combinée à une TD error (équation (17)), pour chaque échantillon, à chaque pas de temps :

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \dots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \quad (29)$$

Avec : $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$

Le principe de l'algorithme est alors le même que celui du A3C, avec N agents en parallèles, qui collectent T pas de temps de données grâce à leurs interactions avec leur propre copie de l'environnement. Avec ces NT pas de temps de données, la fonction de coût substituée est construite, et est optimisée avec un SGD mini-batch, pour K epochs. Le pseudocode de l'algorithme Proximal Policy Optimization est disponible figure 20 à droite.

L'algorithme PPO a été choisi pour son utilisation dans le Use Case 3 du projet KARlab. D'autres algorithmes d'apprentissage par renforcement "état de l'art" ont été étudiés et ne seront décrits que brièvement pour ne pas alourdir le rapport.

9.2 Autres algorithmes états de l'art et limites des méthodes d'AR

9.2.1 Deep Deterministic Gradient Policy (DDPG)

Le DDPG [13] permet d'étendre le Deep Q-Learning (8.2.3) au cas continu. Il est combiné avec le modèle Actor-Critic du Deterministic Gradient Policy (DPG) [12]. Le modèle DDPG utilise deux réseaux de neurones cibles pour représenter la politique déterministe et la valeur Q . Le DDPG utilise aussi l'*expérience replay* pour réduire la corrélation entre les échantillons. Un échantillonnage en mini-batch normalisé est effectué durant l'entraînement.

Fonctionnement : Le principe du DDPG est de maintenir une fonction Actor paramétrisée $\mu(s|\theta^\mu)$ qui spécifie la politique actuelle, en associant de manière déterministe, chaque état à une action. Le Critic $Q(s, a)$ est appris en utilisant l'équation (2) de Bellman du Q-Learning. Pour chaque pas de temps, l'Actor et le Critic (réseaux μ et Q) sont mis à jour, en se basant sur les réseaux cibles μ' et Q' , et en échantillonnant de manière uniforme un mini-batch normalisé depuis un buffer où sont stockées les expériences. De cette manière l'apprentissage se fera à travers un ensemble de données dé-correlées et de même échelle (moyenne et variance

unitaires). Les réseaux cibles eux, permettent de calculer la Q value cible et sont mis à jour de manière douce avec une très petite contribution des réseaux μ et Q , permettant une stabilité d'apprentissage. L'exploration est traité indépendamment de l'algorithme en ajoutant un bruit blanc à la politique d'Actor. La Figure 21 suivante illustre ces propos avec le pseudocode du papier à gauche, et un schéma de [14] à droite :

Algorithm 1 DDPG algorithm

```

Randomly initialize critic network  $Q(s, a|\theta^Q)$  and actor  $\mu(s|\theta^\mu)$  with weights  $\theta^Q$  and  $\theta^\mu$ .
Initialize target network  $Q'$  and  $\mu'$  with weights  $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$ 
Initialize replay buffer  $R$ 
for episode = 1, M do
    Initialize a random process  $\mathcal{N}$  for action exploration
    Receive initial observation state  $s_1$ 
    for t = 1, T do
        Select action  $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$  according to the current policy and exploration noise
        Execute action  $a_t$  and observe reward  $r_t$  and observe new state  $s_{t+1}$ 
        Store transition  $(s_t, a_t, r_t, s_{t+1})$  in  $R$ 
        Sample a random minibatch of  $N$  transitions  $(s_i, a_i, r_i, s_{i+1})$  from  $R$ 
        Set  $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$ 
        Update critic by minimizing the loss:  $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$ 
        Update the actor policy using the sampled policy gradient:
            
$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

        Update the target networks:
            
$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$$

            
$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$

    end for
end for

```

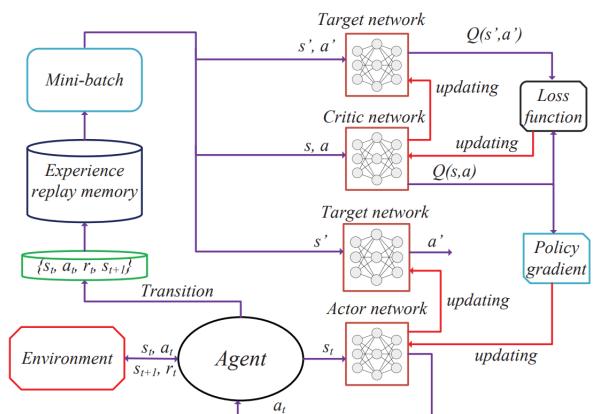


FIGURE 21 – Pseudocode de l'algorithme DDPG (gauche) et schéma de son architecture (droite)

9.2.2 Soft Actor Critic (SAC)

L'algorithme Soft Actor-Critic (SAC) [15] incorpore la mesure de l'entropie de la politique dans la récompense pour encourager l'exploration [16]. On parle aussi d'apprentissage *off-policy* à "entropie maximale". Le but est d'apprendre une politique qui agit de manière aussi aléatoire que possible durant l'entraînement, tout en étant capable de réussir la tâche. La stabilité de l'apprentissage et de l'exploration est assurée par la **régularisation de l'entropie** de cette méthode. De plus le papier [15] propose une incrémentation douce de la politique, adaptée aux problèmes continus.

Fonctionnement : $V_\phi(s_t)$ est défini comme fonction de valeur d'état paramétrée et $Q_\theta(s_t, a_t)$ comme "soft" Q-fonction. $\pi_\phi(a_t|s_t)$ représente la politique. Les fonctions de valeur peuvent être modélisées comme des réseaux neuronaux et la politique comme une gaussienne avec une moyenne et une covariance données par un réseau de neurones. La *soft value* minimise l'erreur quadratique résiduelle et la *soft Q-fonction* minimise l'équation de Bellman résiduelle. Le SAC met à jour les paramètres de la politique, minimisant la DKL et maximisant l'entropie et le retour moyen espéré. L'algorithme utilise aussi un réseau de valeur d'état cible $\bar{V}_\phi(s_t)$ pour stabiliser l'entraînement, ainsi que deux autres fonctions Q pour optimiser le gradient $J_Q(\theta_i)$. Le pseudocode de cette méthode est le suivant, Figure 22 :

Note : Comme le précise [17], l'algorithme SAC a changé avec le temps. Une version moderne omet l'apprentissage de la fonction de valeur $V_\phi(s_t)$.

Algorithm 1 Soft Actor-Critic

```

Initialize parameter vectors  $\psi, \bar{\psi}, \theta, \phi$ .
for each iteration do
    for each environment step do
         $a_t \sim \pi_\phi(a_t | s_t)$ 
         $s_{t+1} \sim p(s_{t+1} | s_t, a_t)$ 
         $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$ 
    end for
    for each gradient step do
         $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$ 
         $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$  for  $i \in \{1, 2\}$ 
         $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ 
         $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$ 
    end for
end for

```

FIGURE 22 – Pseudocode de l'algorithme SAC

9.2.3 Limites des techniques d'AR

Lors de l'utilisation d'algorithmes d'apprentissage par renforcement certaines limites et considérations sont à prendre en compte.

Observabilité Partielle (PO) : Dans beaucoup d'applications réelles, l'agent ne peut pas observer directement l'état réel dans lequel il se trouve, et dispose seulement d'une observation. Celle-ci peut s'éloigner de la réalité dans le cas où certaines variables d'états sont mesurées par des capteurs, qui peuvent ajouter du bruit et un manque de précision. Cette situation est appelée un **Processus de Markov Décisionnel Partiellement Observable (POMDP)**. Une observation peut être définie comme la probabilité d'observer un état Z sachant qu'on se situe sur l'état s' , et que l'action a a été choisie. La fonction d'observation devient donc : $O : A \times S \times Z \rightarrow [0, 1]$.

Fléau de la dimension : Certains problèmes possédant des espaces d'état et d'action continus, génère un nombre d'états et de transitions important. L'apprentissage de l'agent peut donc être long et c'est aussi pour cela que les fonctions d'états et d'états-actions doivent être approchées (Communément avec un réseau de neurones), sinon le stockage d'information évoluerait de manière exponentielle.

Exploration vs. Exploitation : Dans toutes les méthodes présentées précédemment, le but de l'agent est d'apprendre une action optimale pour chaque état possible, de manière à maximiser le gain moyen final lors d'une expérience. Il est donc nécessaire que l'agent parcourt (presque) tous les états possibles, quitte à obtenir quelques fois de moins bon résultats, pour au final ne pas rester bloqué dans une politique sous-optimal (courbe verte figure 23 suivante). Plusieurs méthodes existent donc (ϵ -greedy par exemple) pour trouver un compromis entre exploration et exploitation durant l'apprentissage. Ceci est expliqué sur la figure 23 suivante, pour un problème de "Multi-Armed-Bandit" de [19] :

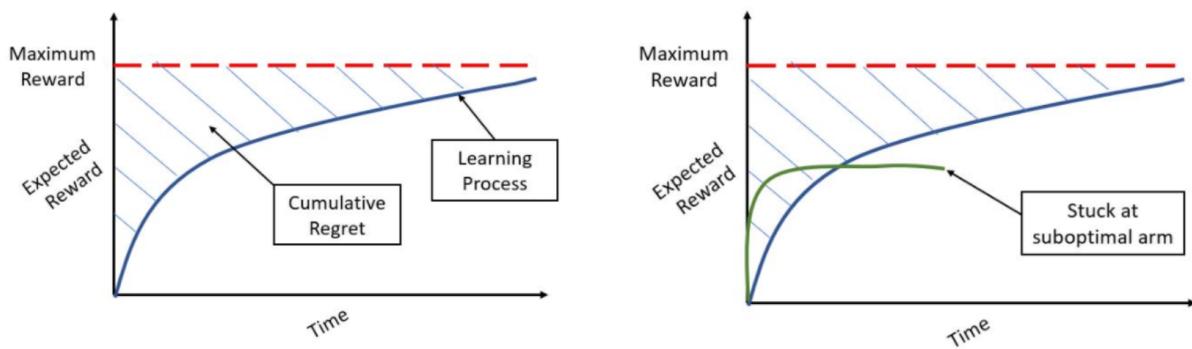


FIGURE 23 – Schéma de la courbe d'apprentissage d'un agent avec et sans utilisation de l'exploration durant son apprentissage.

Corrélation entre les échantillons au cours de l'apprentissage : Un MDP signifie qu'un état donné ne dépend que de l'état de l'instant d'avant. Néanmoins, selon le domaine d'application, un plus grand nombre

d'états peuvent être plus ou moins corrélés. L'introduction d'une «expérience replay», permet à l'agent de rejouer aléatoirement certains tuples d'expérience (état, action, récompense, état-suivant) déjà vécus, au cours d'un épisode. Cela permet d'améliorer l'entraînement et de dé-corréler les échantillons. Dans certains modèles il est même possible de prioriser certaines expériences à rejouer par rapport à d'autres.

Design de la fonction de récompense et tuning des hyperparamètres : Comme mentionné dans l'introduction, dans un problème d'apprentissage par renforcement, la fonction de récompense de l'agent n'est initialement pas connue. Il n'existe pas de solution exacte et son design dépend des exigences que l'agent doit satisfaire lors de la réalisation de sa tâche. Dans le papier [20], les auteurs montrent, en réalisant un benchmark sur différents algorithmes d'apprentissage par renforcement, que les résultats obtenus pour la récompense moyenne obtenue par l'agent, peuvent être peu ou beaucoup influencés par l'échelle de la fonction de récompense. Il est montré également que dans le cas d'une approximation de la politique π et de la fonction de valeur V par réseaux de neurones, le choix des hyperparamètres de ces derniers est parfois crucial dans le résultat final.

Multi-Objectifs (MO) : Dans les problèmes de la vie réel, on a souvent plusieurs objectifs à optimiser et à prendre en compte. Par exemple, pour un trajet en voiture (objectif principal), il est bon d'optimiser à la fois le temps de trajet, son coût, ainsi que la sécurité et le confort des passagers (sous-objectifs). La priorité au sein de ces sous-objectifs est définie par la préférence de l'utilisateur final. Pour une même tâche, l'agent a donc plusieurs comportements complexes à apprendre et à optimiser, avec une ou plusieurs fonctions de récompenses possibles. Le papier [18] propose une *fonction d'utilité* permettant de donner une importance à chacun de ces sous-objectifs.

Multi-Agents (MA) :

Il est possible d'utiliser plusieurs agents pour résoudre un problème d'Apprentissage par Renforcement, et accélérer l'obtention de la politique optimale. Cela peut être nécessaire dans certains problèmes complexes et stochastiques. Dans ce cas, plusieurs agents sont déployés dans un environnement commun et peuvent travailler en collaboration, en compétition, ou les deux. Dans le premier cas, tous les agents peuvent recevoir la même récompense d'équipe. Dans le cas compétitif, les récompenses individuelles peuvent être différentes, et le gain pour un agent peut représenter la perte pour un autre. Néanmoins, ces méthodes peuvent s'avérer très complexes car les agents doivent pouvoir communiquer entre eux, se coordonner et gérer leurs conflits lors de certaines tâches : Les interactions entre les agents changent constamment l'environnement, menant à une non-stationnarité. Dans [18], en prenant en compte les n agents, il est aussi mentionné une modélisation jointe des ensembles S , A et de la fonction de récompense R (potentiellement un ou une pour chaque agent). De plus, chaque agent peut posséder une politique différente.

10 État de l'art du stationnement autonome par AR : Analyse et choix technique

La partie suivante comporte un résumé de l'étude et de la comparaison des méthodes d'AR existantes pour le stationnement autonome et/ou la prise de décision d'un véhicule.

10.1 Analyse de l'existant

Papier 1 : "Controlling an Autonomous Vehicle with Deep Reinforcement Learning" (2019) [21]

Méthode et algorithme utilisés : Deux fonctions de récompense ("Driver" et "Stopper") sont utilisées ; L'espace d'état regroupe une carte de perception, la position, la vitesse et l'orientation du véhicule, ainsi que la position relative de la cible et son orientation ; L'espace d'action correspond à l'accélération et à l'angle des roues : Ces actions sont bornées à des valeurs raisonnables pour que l'agent n'ai pas un comportement trop brusque ; V et π sont approximées par des réseaux de neurones. La perception du véhicule est obtenue avec les capteurs et normes du "AO-CAR" project [22] ; Un Filtre de Kalman Etendu (EKF) est utilisé pour le suivi du

véhicule et l'algorithme **PPO** est utilisé pour l'entraînement de l'agent ; La fonction de récompense donne un retour positif lorsque l'agent atteint la cible ou se rapproche de la cible, et un retour négatif lorsque l'angle des roues ou la vitesse de l'agent est trop importante.

Entraînement, Tests et Résultats : L'entraînement de l'agent est fait sur une carte graphique GTX 1080 GPU et prend une dizaine d'heures. L'inférence du réseau de neurones représentant la politique sur le choix des commandes à actionner sur le véhicule est de 1.2ms sur un CPU Intel Core i7-4790. Au final, le véhicule peut se rendre d'un point à un autre, suivre une trajectoire et prendre une décision s'il doit plutôt éviter ou s'arrêter en présence d'un obstacle. Cette solution s'approche extrêmement du Use Case 3 étudiée, car le véhicule doit se rendre d'un point à un autre. Le résumé et les résultats en **tests réels** de ce papier sont fournis à travers ce [lien Youtube], et une illustration est disponible figure 24.

Papier 2 : "Robust Auto-parking : Reinforcement Learning based Real-time Planning Approach with Domain Template" (2018) [23]

Méthode et algorithme utilisés : Le stationnement est divisé en 3 étapes : se rapprocher, planifier et se garer, puis s'ajuster. La fonction de récompense éparsé pénalise l'agent quand il rentre en collision ou quand l'angle des roues est trop important ; Elle le récompense quand la cible est atteinte ; Un réseau de neurones est utilisé pour représenter la politique et l'apprentissage se fait avec l'algorithme **PPO** et avec *Curriculum Learning* [33] ; Un autre réseau "*Long-Short-Term-Memory*" (LSTM) est aussi testé pour représenter la politique ; Les auteurs comparent leurs résultats qui se révèlent meilleurs que ceux obtenus avec un algorithme classique d'AR, combiné à une planification de trajectoire ("DQfD").

Entraînement, Tests et Résultats : L'entraînement et les tests sont réalisés sur un simulateur "haute fidélité" ("high-fidelity simulator") ; Le véhicule arrive bien à se garer et les résultats obtenus avec la méthode par AR montrent une courbe de trajectoire plus douce qu'avec les méthodes classiques ; Le système est aussi assez robuste au bruit ajouté sur la détection de la place de parking.

Papier 3 : "Reinforcement Learning-Based End-to-End Parking for Automatic Parking System" (2019) [24]

Méthode et algorithme utilisés : Le stationnement est divisé en 2 étapes : se rapprocher de la place de parking, puis stationner en bataille arrière ; 4 caméras classiques d'ADAS permettent au véhicule de détecter les coins des places de parking ; Le suivi de l'agent est assuré par un EKF et une centrale inertiel (IMU) ; La fonction de récompense permet de favoriser le stationnement du véhicule et son alignement avec la place de parking ; Elle pénalise son empiètement sur les lignes de parking. L'espace d'état contient la position et le cap du véhicule, les coins de la place de parking et la distance restante entre les deux. L'action résultante est l'angle des roues. L'algorithme utilisé est le **DDPG**.

Entraînement, Tests et Résultats : Un entraînement en simulation à d'abord lieu, où le véhicule est modélisé avec Matlab et l'environnement de parking avec le logiciel PreScan ; Une optimisation de l'entraînement est faite avec le **Curriculum Learning** et une exploration manuelle. Le **test en conditions réelles** est ensuite réalisé à une vitesse de 4km/h et la méthode par AR obtient finalement une précision d'alignement de $\pm 1^\circ$ par rapport à celui désiré. Une illustration du résultat est disponible figure 25.

Papier 4 : "Reinforcement Learning-Based Motion Planning for Automatic Parking System" (2020) [25]

Méthode et algorithme utilisés : Méthode model-based pour le stationnement en parallèle, avec une politique pour le contrôle longitudinal et une autre pour le contrôle latéral ; Algorithme **MCTS** : génération de données, simulation, évaluation ; Utilisation d'un Lidar 2D et d'ultrasons ; L'espace d'état correspond à la position et au cap du véhicule et à la longueur de la place de parking ; L'espace d'action est composé de l'angle des roues seulement ; La fonction de récompense pénalise les collisions et favorise le confort des passagers.

Entraînement, Tests et Résultats : Un **test réel** est réalisé et obtient une erreur de cap moyenne de -0.5° et une erreur de posture moyenne de -0.83 mètres ; Cette méthode divise le temps de stationnement par 2 par rapport aux approches géométriques classiques. Des illustrations de la méthode sont présentes figure 27.

Papier 5 : "Trajectory Planning for Automated Parking Systems Using Deep Reinforcement Learning" (2020) [26]

Méthode et algorithme utilisés : Cette méthode se base sur les travaux du **DQN** ; L'entrée du système correspond à la vue périphérique interprétée du véhicule, ainsi qu'aux données classiques de perception d'un véhicule (position, capteurs à ultrasons etc.)

Entraînement, Tests et Résultats : L'entraînement est réalisé avec Tensorflow et OpenCV. Il n'y a **pas de tests réels**, seulement une simulation en temps réel ; L'erreur de cap avec la place de parking est d'environ 5° , et l'erreur de position d'environ 0.3 mètres.

Papier 6 : "Self-Parking Car Simulation using Reinforcement Learning Approach for Moderate Complexity Parking Scenario" (2020) [27]

Méthode et algorithme utilisés : Les auteurs modélisent l'environnement de parking automatique sur Unity, pour valider un modèle avec une complexité simplifiée. Le modèle de capteurs à ultrasons Unity est utilisé autour du véhicule. Le plugin ML-Agents et l'algorithme **PPO** sont utilisés ; L'espace d'état correspond aux positions de l'agent et de la zone de parking ainsi qu'aux informations des capteurs. La vitesse et le cap de l'agent sont aussi ajoutés ; L'espace d'action correspond à l'angle des roues et à l'accélération ; La fonction de récompense favorise la rapidité de l'agent ainsi que son alignement avec les places de parking ; Les collisions sont pénalisées.

Entraînement, Tests et Résultats : Le réseau de neurones de la politique est entraîné avec TensorFlow et OpenAI Gym, et plusieurs fonctions de récompenses sont testées ; Finalement le véhicule peut se garer avec 90% de chance de succès sur les places de parking les plus proches "zone A", et seulement 10% sur les places les plus éloignées "zone B". La figure 26 montre l'environnement de parking du simulateur.

Papiers 7 : "Hybrid DDPG Approach for Vehicle Motion Planning" (2019) [28] and "Proving Ground Test of a DDPG-based Vehicle Trajectory Planner" (2020) [29]

Méthode et algorithme utilisés : Modèle de générateur de trajectoire et de contrôle longitudinal et latéral, prenant en entrée l'état initial et final du véhicule ; La fonction de récompense favorise le lissage du mouvement et pénalise l'erreur de cap ou de suivi de trajectoire ; La perception classique ADAS est présente avec la présence d'un GPS, d'une IMU, d'un Radar, d'un Lidar et d'une Caméra ; L'espace d'action est continu et correspond à l'angle des roues, à l'accélération et au freinage ; L'algorithme **DDPG** est utilisé.

Entraînement, Tests et Résultats : L'entraînement est réalisé sur simulateur, et dure de 8h à 20h selon la vitesse à laquelle la tâche doit être effectuée ; Un **test réel** avec une Smart sur le circuit ZalaZone est effectué. La performance sur la manœuvre la plus complexe est un évitement de 7.5 mètres à une vitesse de 40km/h avec une erreur de position moyenne de 0.3 mètres et une erreur de cap de 0.04° .

Papier 8 : "Reverse Parking a Car-Like Mobile Robot with Deep Reinforcement Learning and Preview Control" (2019) [30]

Méthode et algorithme utilisés : L'algorithme utilisé est le **DDPG** en association avec une loi de contrôle ; L'environnement de parking étant modélisé à la verticale, l'espace d'état est représenté par la position x du véhicule et par son cap ϕ . L'espace d'action correspond à l'angle des roues et au contrôle longitudinale.

Entraînement, Tests et Résultats : L'entraînement est réalisé sur Matlab et 4000 épisodes sont nécessaires avant la convergence. L'utilisation de leur méthode permet d'atteindre l'objectif 2.8 secondes plus rapidement que sans la loi de contrôle en amont.

Note :

- Certains papier ne sont pas accessibles directement, et doivent être consultés avec le site "Sci-Hub" (en utilisant un VPN si l'accès est bloqué).
- Beaucoup de publications se rapprochent de la problématique, mais ne traitent pas exactement de stationnement autonome, ne respectent pas le cahier des charges, ne sont pas assez abouties, ou utilisent des modèles trop simplifiés. Un tri a donc été fait. Il est aussi possible que d'autres papiers traitant ou s'approchant de la problématique du Use Case 3 n'aient pas été trouvés ou étudiés.
- L'Apprentissage par Renforcement Profond (Deep Reinforcement Learning), ainsi que son application à l'automobile, étant des domaines actuellement en plein essor, de nouveaux papiers pertinents ont pu être publiés après que cet état de l'art eut été réalisé.

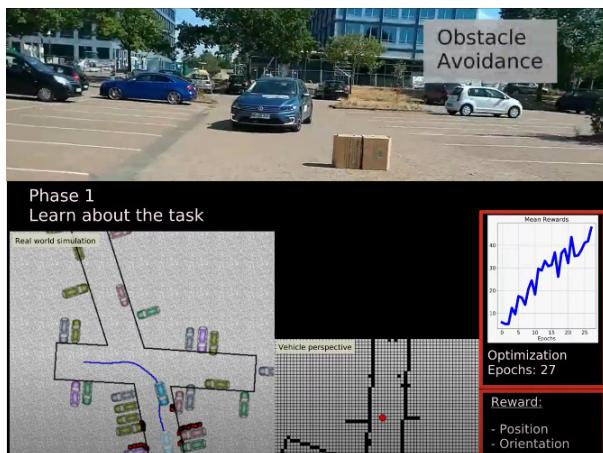


FIGURE 24 – Résultats et entraînement, papier 1

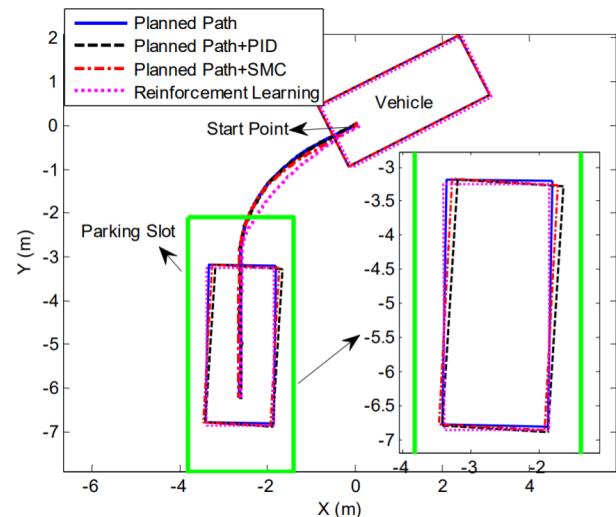


FIGURE 25 – Illustration des résultats, papier 3

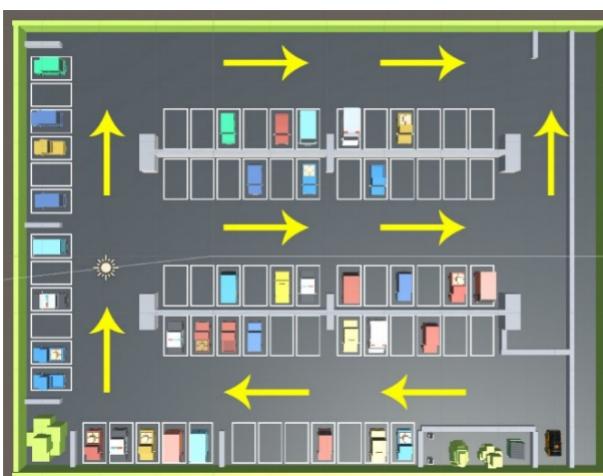


FIGURE 26 – Environnement Unity, papier 6

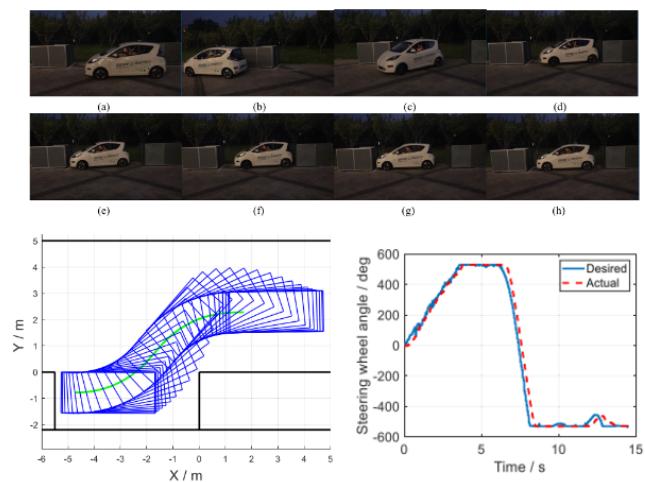


FIGURE 27 – Illustrations des résultats, papier 4

10.2 Discussion sur les papiers étudiés et sur les algorithmes d'AR

Le papier 1 [21] est celui se rapprochant le plus du besoin étudié. Par contre, il ne spécifie pas sur quel simulateur le pré-entraînement a été réalisé. De plus, le frein ne fait pas parti de l'espace d'actions de l'agent : Soit il s'arrête, soit il contourne l'obstacle mais ne ralenti pas pour repartir ensuite. Néanmoins, la vidéo de

l'entraînement sur simulateur et celle du test dans la vie réelle, figure 24, sont des preuves conséquentes de son efficacité et de sa fonctionnalité.

Le papier 2 [23], obtient des résultats intéressants au niveau du "*smoothing*" de la trajectoire de stationnement. L'algorithme PPO est utilisé. Dans le cas de la représentation de la politique par un LSTM, cela peut complexifier le modèle et la stabilité de l'apprentissage. De plus, le système est robuste seulement pour fonctionner à proximité de la place de parking. Seul l'angle des roues est présent dans l'espace d'actions.

Dans le papier 3 [24], 4 caméras sont utilisées pour avoir une vision à 360° et détecter les coins des places de parking. Le Use Case 3 de KARlab ne spécifie pas ces caractéristiques. Aussi, un test sur véhicule réel obtient un très bon résultat final, figure 25. L'agent est entraîné par DDPG pour choisir seulement l'angle des roues "*steering wheels angle*" à chaque instant. De même, le papier 4 [25], qui utilise un algorithme "*Monte-Carlo Tree Search*" pour une planification de trajectoire, figure 27, ne permet d'obtenir que l'angle des roues en sortie du système. On ne peut donc pas parler de stationnement purement autonome pour les trois papiers précédents.

Le papier 5 [26] s'éloigne aussi en partie de la problématique, car nécessite une détection de la place de parking, pour générer ensuite une planification de trajectoire. Cela n'est pas possible dans le cas investigué, puisque l'agent doit pouvoir réagir à la présence d'un éventuel obstacle ou piéton survenant sur sa route. La méthode n'est pas non plus testée dans la vie réelle, et discrétise l'espace d'action, représenté seulement par l'angle des roues.

Le papier 6 [27] est proche du système recherché puisque le véhicule parvient à se garer seul avec alignement, en agissant sur l'angle des roues et l'accélération. Bien que l'agent évolue dans un environnement virtuel et simplifié (figure 26). , cela reste réaliste et permet une première projection du système, avant son transfert dans la vie réelle. L'agent n'utilise pas de frein, et n'est pas entraîné pour éviter un obstacle sur sa route. Les résultats sont aussi moins bons lorsque la distance entre l'objectif et le véhicule augmente.

Les papiers 7 [28] et [29], mettent en évidence des résultats en tests réels concluants, grâce à l'utilisation de l'algorithme DDPG, mais réalisent aussi une planification de trajectoire.

Le papier 8 [30] utilise seulement une simulation Matlab très simplifiée avec la technique du DDPG.

10.2.1 Limites des papiers de l'état de l'art

Les papiers ne détaillent pas comment le véhicule s'arrête sur la place de parking. Il ne traitent pas toujours de la vitesse de l'agent aux abords de la place de parking, ni de son utilisation du frein. Il n'est pas précisé si l'épisode est réussi quand l'agent est strictement arrêté sur la place de parking, ou si il est considéré comme réussi dès que l'agent et la place de parking partagent environ la même position. Les papiers bornent ou fixent la vitesse de l'agent quand elle est mentionnée. Les moyens et logiciels de simulations ne sont pas toujours détaillés.

Les travaux effectués ne respectent pas toujours la normalisation de l'espace d'état et celui de la fonction de récompense, comme recommandée dans d'autres sources [16]. De plus la fonction de récompense n'est pas toujours testée avec différents designs. Le tuning des hyperparamètres n'est pas non plus mentionné, ce qui peut être compréhensible, de part la longueur de l'apprentissage d'un agent dans un cas d'AR.

Pour finir, les papiers traitant seulement d'algorithmes d'AR ne comparent pas toujours ces derniers avec tous les autres algorithmes d'AR, ou au moins avec les algorithmes "état de l'art". Les tâches à réalisées ou les environnements (OpenAI Gym, MuJoCo) et conditions d'entraînements peuvent être parfois trop simplifiées et non uniformisées lors des tests. Cela peut rendre la comparaison des performances difficile.

10.2.2 Similitudes des papiers de l'état de l'art

Plusieurs papiers utilisent le Curriculum Learning dans le but de maximiser les chances de convergence vers une politique optimale et accélérer le temps d'apprentissage. Les papiers testant leur méthode sur un véhicule réel, entraînent d'abord l'agent sur un simulateur, pour des questions de coût, de temps et de sécurité. Le modèle

doit être également légèrement ré-entraîné et ajusté lors de son utilisation dans l'environnement réel. L'espace d'états des systèmes contient toujours plus ou moins les mêmes informations : L'état du véhicule (position, cap, vitesse), l'état de la place de parking, une information de distance entre les deux, et les informations provenant des capteurs ou caméras du système. L'espace d'action est constitué de maximum 3 composantes : l'accélération, l'angle des roues et le freinage.

Des réseaux de neurones sont utilisés pour approximer les fonctions d'AR \mathbf{V} , \mathbf{Q} ou π . La fonction de récompense assure la bonne réussite de l'objectif, mais aussi le lissage du comportement du véhicule et la sécurité du modèle. La fonction de récompense et l'espace d'états sont de préférence normalisés, de manière à stabiliser l'apprentissage.

10.3 Solution envisagée

Dans ces papiers décrits précédemment, ceux qui se rapprochent le plus du besoin initial, utilisent l'algorithme PPO, les autres le DDPG. Après d'autres recherches additionnelles, des papiers comme [34], proposent une étude comparative détaillée des algorithmes d'AR. Les auteurs montrent que les performances d'un algorithme dépendent aussi beaucoup de l'implémentation qui en est faite. Le taux de réussite peut être plus ou moins similaire, et peut varier en fonction du problème traité. Néanmoins, l'algorithme PPO paraît plus simple d'implémentation et plus stable durant l'apprentissage. Il est utilisé dans beaucoup de framework de Deep Reinforcement Learning. C'est aussi l'algorithme le plus cité par rapport à son nombre d'années d'existence. D'après [35], le DDPG peut converger plus vite sur certaines tâches, mais possède un apprentissage plus instable. L'algorithme Proximal Policy Optimisation a donc été choisi bien que nous gardions quand même l'idée de tester le DDPG pour comparer les performances de chacun.

Pour l'environnement de test, l'idée de départ était d'utiliser le logiciel open source "*CARLA Simulator*" [32], car celui-ci permet une assez bonne approximation de la réalité. Néanmoins, il nécessite des ressources graphiques conséquentes, car se base sur le moteur de jeux vidéos *Unreal Engine*. Dans le cas d'une première Proof of Concept (PoC), cela peut allonger considérablement le temps de développement, d'entraînement, de *tuning* et de validation d'un modèle. De plus, aucun travail de stationnement autonome, ou se rapprochant du Use Case 3 n'a été trouvé sur Carla. Le papier de CARLA [32] spécifie que les créateurs du simulateur ont entraîné un modèle de conduite autonome en ville avec l'algorithme A3C. Ils ont entraîné "10 agents en parallèle pour un total de 10 millions de steps. [...]. Cela correspond à environ 10 images par secondes pendant 12 jours. Au final, ils n'ont obtenu qu'un résultat de 65% de réussite lors des tests.

Les deux papiers qui se rapprochent le plus de la problématique sont les numéros 1 et 6. Le premier ne spécifie pas son simulateur et le second utilise Unity. Il est donc possible de s'inspirer du papier 6 [27] en l'améliorant et le ré-appropriant à KARlab. Une première PoC serait développée ainsi qu'une première validation d'un système de stationnement autonome par AR. De plus, même si moins formalisés, d'autres exemples de projets de la sorte ont été réalisés avec le même framework, et rencontrés sur les liens suivants (lien 1, lien 2, lien 3). Le plugin ML-Agents [31] dispose également d'une documentation conséquente pour développer ses propres modèles d'AR efficacement. Les mécanismes d'entraînement de l'AR sont assez détaillés.

À l'issue de l'implémentation sur Unity, d'un choix et d'une validation du meilleur modèle, ce résultat peut être repris pour finaliser son entraînement avec CARLA Simulator. Ou bien, le même modèle d'AR peut être modélisé de nouveau sur cet autre simulateur afin de ré-entraîner l'agent avec une modélisation plus complexe.

11 Réalisations

11.1 Modélisation du stationnement autonome par AR sur Unity

Unity est un moteur de jeux vidéo permettant de modéliser en 3 dimensions une "scène" qui s'approche d'une réalité visuel et physique (Pokemon go et Among us ont notamment été créés dessus). Ce logiciel utilise NVIDIA PhysX, un moteur de physique puissant, permettant de bénéficier d'effets physiques en temps réel pour ses simulations.

L'intégralité du gameplay et de l'interactivité développée dans Unity est basée sur trois blocs de base : les *GameObjects*, les *composants* et les *variables*. Tout objet Unity est un *GameObject* : c'est le cas des personnages, des éclairages, des effets spéciaux, des accessoires, etc. Les composants permettent le contrôle du comportement des *GameObjects* auxquels ils sont attribués. Les composants ont de nombreuses propriétés modifiables, ou variables. Elles peuvent être ajustées directement dans l'Éditeur Unity et/ou via un script, pour appliquer une logique et des comportements propres désirés. Un simple exemple est la création d'un *GameObject* représentant un véhicule, auquel on va ajouter un composant "*Rigidbody*" pour qu'il puisse avoir un certain poids et interagir avec le sol.

Les scripts Unity utilisent le langage C#, qui est un langage de programmation orienté objet, commercialisé par Microsoft depuis 2002. C'est un langage dérivé du C++ qui est très proche du Java dont il reprend la syntaxe générale, ainsi que les concepts.

11.1.1 Modélisation

Modélisation de l'agent

Le kart est modélisé avec les mêmes dimensions (poids, taille, rayon et taille des roues etc.) que celles du kart réel du projet KARlab : Sodikart RSX 04, figure 2. Le véhicule est l'agent, et sa physique est modélisée sur Unity en se servant de la documentation "*WheelColliders*" [36]. L'agent dispose alors d'un châssis, matérialisé par un "*Box Collider*" figure 28, et de 4 "*Wheel Colliders*" modélisant les roues. Les roues peuvent changer de direction et freiner. Un "*Torque*" (couple moteur en français) est appliqué sur ces dernières pour faire avancer le véhicule. Un script permettant l'AR du stationnement autonome est attaché à l'agent pour qu'il puisse interagir avec son environnement.

Le projet KARlab dispose aussi d'un radar, qui spécifie une observation du plan azimutale du kart de $\pm 60^\circ$, avec une résolution de 15° entre deux objets, et une portée maximale de 80 mètres. Tous ces paramètres sont disponibles grâce au composant "*Raycast Observations*" de Unity, qui est donc attaché au kart pour modéliser le radar. Un raycast est un laser partant du centre du kart. Il détecte la distance à laquelle il touche un objet et peut retourner le type d'objet détecté.

Ci-dessous à gauche, figure 28, se trouve la modélisation physique du kart, sans le visuel associé : Les lignes vertes représentent le *Box Collider* qui est aussi la répartition de masse du kart. Les cercles verts correspondent à la modélisation des roues. Il est aussi possible d'apercevoir les axes du repère local au kart, ainsi que la modélisation d'obstacles (blocs blancs), et d'un piéton sur la zone de parking. Les 9 lignes rouges représentent la modélisation du radar dont dispose l'agent.

Modélisation de l'environnement

L'environnement a été créé en s'inspirant d'un projet personnel de stationnement autonome par AR de Valecillos D. [37], et en se basant sur le "*Hello World*" basique du plugin [39], où une sphère doit atteindre la position un objectif. L'environnement d'AR pour le kart consiste donc en une zone de parking, qui contient le kart décrit précédemment, ainsi qu'une place de parking qui lui est associée. Ci-dessous à droite, figure 28, se trouve une partie de l'environnement modélisé. La place de parking est aperçue : Elle est l'objectif que doit atteindre l'agent. Sa modélisation est faite par 3 lignes blanches avec un carré vert au centre. Sur cette image les blocs blancs correspondent aux obstacles et l'humanoïde à un piéton.

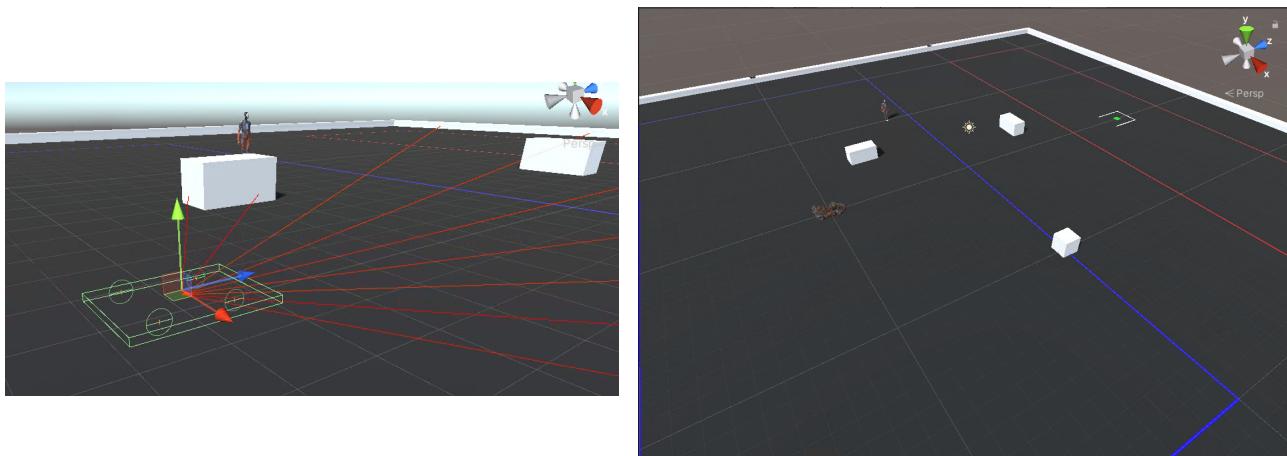


FIGURE 28 – Modélisation Unity de l’agent et de son environnement

11.1.2 Unity ML-Agents

ML-Agents est un projet open-source qui permet aux jeux et aux simulations Unity de servir d’environnements pour l’apprentissage d’agents intelligents. Des implémentations, basées sur PyTorch, d’algorithmes de pointe s’interfacent avec l’API Python fournie par le package. Les développeurs peuvent alors entraîner des agents intelligents pour des jeux en 3D et en VR/AR. La plate-forme est aussi utile aux chercheurs en IA qui peuvent l’utiliser pour entraîner des agents avec l’apprentissage par renforcement, auquel d’autres méthodes peuvent être ajoutées (*GAIL*, *Behavior Cloning* etc.). ML-Agents constitue une plate-forme centrale où les avancées en IA peuvent être évaluées dans les environnements riches d’Unity.

La mise en place de ce plugin consiste à télécharger le code source "ml-agents" sur GitHub, créer un environnement python avec les librairies nécessaires au plugin, puis télécharger deux librairies dans l’éditeur Unity. Anaconda, le gestionnaire d’environnements Python, est donc utilisé par commodité.

Ce plugin permet de convertir n’importe quelle scène Unity en un environnement d’apprentissage et apprendre un comportement à un agent. Une fois entraîné, le modèle formé peut alors être réintégré dans la scène Unity. Les méthodes de la librairie peuvent être redéfinies afin de les utiliser dans l’environnement Unity souhaité et organiser l’apprentissage. Ce sont les suivantes :

- `OnEpisodeBegin()` permet de réinitialiser la scène, et de définir les conditions initiales d’un épisode : La position de l’agent, celle de la place de parking etc.
- `CollectObservations()` est utilisée pour chaque agent de la scène, pour mettre à jour ses observations (positions, raycasts, vitesse, etc).
- `OnActionReceived()` permet de collecter pour chaque agent de la scène l’action à effectuer. Cette collecte peut-être dictée par le clavier du développeur ("*Heuristic mode*"), pour tester le fonctionnement de l’environnement, par l’API Python, lors de l’entraînement de l’agent, ou bien par le modèle de réseau de neurones lors d’un test d’inférence ("*Inférence mode*").

L’entraînement se fait alors de manière externe à Unity avec l’API Python, qui collecte les données des épisodes, de manière à optimiser le réseau de neurones représentant la politique du comportement à apprendre. De plus, un des grands avantages du plugin ML-Agents, est qu’il permet de paralléliser les environnements d’apprentissage, dont les expériences alimentent un *buffer* d’expériences. Ces dernières sont beaucoup plus décorrélées qu’avec un seul agent, et utilisées aléatoirement : cela permet d’explorer plus d’états différentes, et donc d’accélérer et de stabiliser l’apprentissage. À gauche figure 29 ci-dessous, se trouve un schéma du fonctionnement de ML-Agent. L’image de droite correspond au rendu visuel lors du clonage de l’environnement créé.

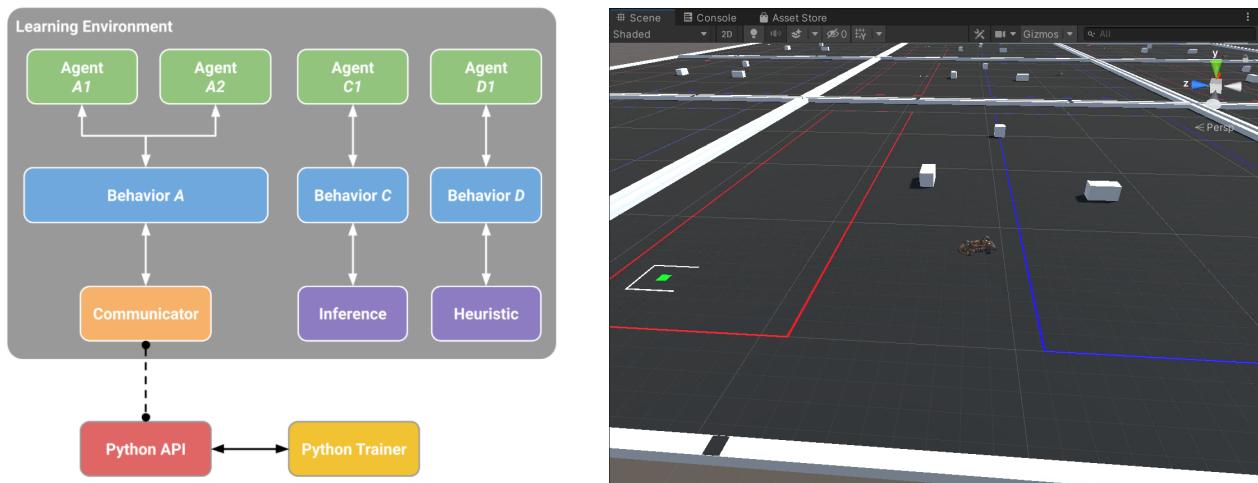


FIGURE 29 – Schéma du système du plugin ML-Agents et illustration de son utilisation pour le Use Case 3

Cette parallélisation permet à chaque agent d'évoluer dans le même environnement de parking. Ils ont tous le même objectif : Se garer en s'alignant avec la place de parking, tout en évitant les obstacles. Lors de l'entraînement, les tuples (état, action, récompense, état-suivant) de chaque agent, sont collectés et envoyés à L'API Python externe à Unity. Cette dernière utilise l'algorithme choisi et précisé en paramètre, pour effectuer la descente de gradient et renvoyer les actions que l'agent doit effectuer. Un exemple de ces communications est schématisé sur la figure 29 précédente.

11.1.3 Entrainements et tests

Modèle d'AR utilisé et modalités

Le système prend en entrée l'espace d'états de l'agent, qui sont ses observations. Il est alors entraîné pour que, à chaque instant, il associe à l'état observé la meilleure action possible, de manière à se rapprocher de son objectif, et à maximiser le retour moyen espéré.

Le script servant pour l'AR qui est attaché à l'agent, précise que 3 valeurs de type "float" sont obtenues en sortie du système, et sont comprises respectivement entre $[-1; 1]$, $[-1; 1]$ et $[0; 1]$. Ces valeurs peuvent être vues comme un pourcentage, et sont apprises par l'agent. Elles sont ensuite multipliées respectivement au `maxMotorTorque`, au `maxSteeringAngle` et au `maxBrakeTorque` du kart modélisé sur Unity. Le véhicule recevra donc au final 3 variables d'actions, appelées `motor`, `steering` et `brake`.

Lors du lancement de la commande d'un entraînement dans le terminal, il est nécessaire de spécifier en argument le chemin d'un fichier au format ".yaml", qui contient l'algorithme choisi et tous ses hyperparamètres à utiliser durant l'entraînement. Une fois la commande exécutée, on peut voir sur cette [VIDEO 1], le début de l'entraînement d'un agent, qui avance et recule aléatoirement pour tester différents états. Dans Unity, le rendu visuel est accéléré.

PyTorch est une bibliothèque open source pour effectuer des calculs à partir de graphs de flux de données et de modèles de deep learning. La librairie ML-Agent permet de choisir entre l'algorithme PPO et l'algorithme SAC de PyTorch, pour l'optimisation de la politique. Ils sont déjà implémentés et reliés à l'agent à entraîner, lorsque l'on connecte L'API Python et Unity pour l'entraînement.

Il a été décidé de tester et de comparer les performances des algorithmes PPO et SAC pour ce modèle d'AR. L'idée du test du DDPG, algorithme visiblement instable et complexe à implémenter "*from scratch*" d'après 21, et potentiellement pas possible d'inclure dans un environnement Unity ML-Agent, a été abandonnée.

Le résultat d'un entraînement est un fichier au format open source ".onnx" [40]. Ce format assure l'intercompatibilité entre les modèles d'IA et les différents frameworks associés (PyTorch, TensorFlow, etc.). Une fois l'entraînement terminé et la politique optimisée, il est possible d'associer le modèle .onnx obtenu, à un agent, pour réaliser l'inférence dans l'environnement Unity développé, et tester la précision du modèle obtenu.

Lors d'un entraînement, les statistiques de l'apprentissage sont sauvegardées : Tensorboard, outil développé par TensorFlow [41], permet de visualiser l'évolution de tous les hyperparamètres et autres paramètres de l'entraînement d'un modèle. Dans ce rapport, c'est surtout le retour moyen espéré qui sert de comparaison de performance et de mesure de temps de convergence, de manière à ne pas détailler à chaque fois tous les hyperparamètres.

Configurations testées

Des premières sessions d'entraînement de l'agent sont menées, avec d'abord une configuration par défaut du fichier `.yaml` (avec l'algorithme PPO), de manière à trouver le meilleur espace d'états et la meilleure fonction de récompense pour l'apprentissage de l'agent. Le Use Case 3 étudié est aussi simplifié, en essayant d'abord une politique sur le **scénario 1**, défini comme suit :

"La place de parking se trouve à moins de 30 mètres de l'agent, le frein n'est pas encore mis en place dans le système, et le véhicule ne gère pas la vitesse à laquelle il arrive sur la place de parking. L'agent et la place de parking sont chacun initialisés à une position aléatoire dans la zone de parking. L'épisode est terminé lorsque l'agent s'éloigne de plus de 30 mètres du parking, dépasse la limite de temps, ou réussi à se garer (distance au parking inférieur à 0.5 mètres)".

Le premier succès a alors été obtenu avec la **configuration 1** suivante :

- Espace d'états : L'angle de cap de l'agent en degré (1 valeur) ; sa position (x,y,z) (3 valeurs) ; ses composantes de vitesses dans le plan (2 valeurs) ; la position de la place de parking (3 valeurs).
- Fonction de récompense : -1 si la distance entre l'agent et la place de parking dépasse 30 mètres ; -1 si l'agent dépasse le temps imparti (environ 1 minute) ; $\frac{-1}{\text{MaxSteps}}$ à chaque instant pour inciter l'agent à se garer rapidement ; $+1$ si la distance entre l'agent est la place de parking est inférieur à 0.5 mètres.

Bien que ce modèle ait pu appris en 1 heure, il ne respectait pas l'alignement avec la place de parking, et allait se garer en faisant des cercles en marche arrière comme sur cette [\[VIDEO 2\]](#). Cette première PoC n'incluait pas non plus la présence d'obstacles. L'annexe 15 désigne les premiers hyperparamètres testés par défaut.

Pour optimiser le système, les recommandations de la documentation du plugin et d'autres sur l'entraînement d'un modèle d'AR ont été suivies pour l'optimisation du modèle : Augmenter le nombre d'agents travaillant en parallèle permet de collecter beaucoup plus d'expériences différentes, explorer beaucoup plus d'états, et d'accélérer l'entraînement, figure 30 à gauche ; Pénaliser de manière plus importante l'agent lorsqu'il n'est pas aligné, permet d'obtenir un meilleur résultat final, figure 30 à droite ; Normaliser les observations permet d'accélérer l'apprentissage, figure 33 à gauche.

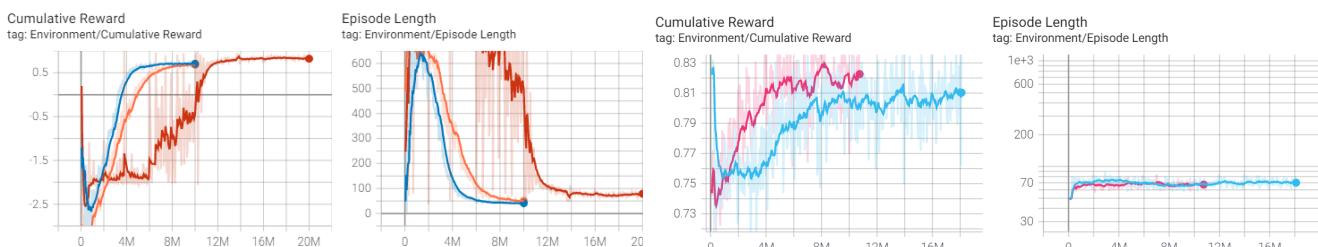


FIGURE 30 – Comparaison de l'entraînement du modèle d'AR en faisant varier le nombre d'agents en parallèles, gauche, et en faisant varier la fonction de récompense, droite

La figure 30 précédente à gauche, montre que l'augmentation du nombre d'agents diminue le temps d'entraînement : La courbe rouge correspond à l'évolution de la récompense moyenne au cours du temps avec 8 agents travaillant en parallèle, sur un PC portable de l'entreprise. La courbe orange est obtenue avec 32 agents sur le PC du labo KARlab avec une meilleure configuration. La courbe bleu est celle de l'entraînement avec 256 agents en parallèle sur le même PC. Les temps respectifs de convergence sont de 4h30, 1h50 et 1h30.

Sur la même figure image de droite, la courbe bleue représente la récompense moyenne de l'agent au cours du temps, en lui appliquant une pénalité de -0.1 si l'erreur d'alignement est supérieur à $\pm 10^\circ$. Celle en rose représente une pénalité de -0.5 . Il semble qu'au début de l'apprentissage la récompense totale moyenne espérée est plus élevée si une pénalité de -0.1 est appliquée pour l'alignement, mais qu'à terme, la convergence et la performance sont meilleures dans le cas d'une pénalité plus élevée.

Après de nouvelles optimisations, une PoC plus avancée de stationnement autonome sans obstacle, avec alignement, à une distance inférieure à 30 mètres et sans gérer l'arrêt sur la place de parking, a donc été obtenue avec la **configuration 2** suivante :

- Espace d'états : L'angle de cap de l'agent en degré ; sa position ; ses composantes de vitesses dans le plan ; la position de la place de parking ; **Les composantes en 3D de la distance restante entre la place de parking et l'agent** ; **Toutes les observations sont normalisées**.
- Fonction de récompense : -1 si la distance entre l'agent et la place de parking dépasse 30 mètres ; -1 si l'agent dépasse le temps de parking imparti (environ 1 minute) ; $\frac{-1}{\text{MaxSteps}}$ à chaque instant pour inciter l'agent à se garer rapidement ; -0.001 à chaque instant si l'agent utilise une action de torque négative, pour l'éviter de reculer ; -0.001 à chaque instant si l'agent utilise une action faisant tourner ses roues avec un angle de supérieur à $\pm 15^\circ$, pour l'éviter de trop tourner et garder un déplacement fluide ; -0.5 si l'agent a une erreur d'alignement avec la place de parking de $\pm 10^\circ$; $+1$ si la distance entre l'agent et la place de parking est inférieur à 0.5 mètres.

Lors du test d'un agent dans cet environnement de parking avec la politique apprise, on a 98% (2165/2209 d'épisodes réussis) de réussite de parking de l'agent, et 97,5% (2154/2209) de réussite de parking avec alignement. Cette inférence est disponible [VIDEO 3].

Après avoir entraîné un modèle approximant grandement les spécificités visées, le "tuning" des hyperparamètres a été réalisé. Étant nombreux et l'entraînement d'un agent pouvant être long (de plusieurs heures jusqu'à plusieurs jours), les différentes valeurs des hyperparamètres ont été testées sur un **scénario simple**, permettant une convergence plus rapide : *"La position de l'agent et celle de la place de parking sont toujours initialisées au même endroit : ils sont situés l'un en face de l'autre, à 5 mètres de distance"*.

Les algorithmes PPO, SAC et POCA fournis par le plugin sont d'abord testés. Le dernier cité est une méthode d'AR développée par l'équipe ML-Agent, qui peut être aussi utilisée dans un contexte multi-agent. La release du plugin actuel, ne comporte pas encore de documentation officielle de cet algorithme. Sur le scénario simple, les résultats sont les suivants pour le PPO, SAC et POCA :

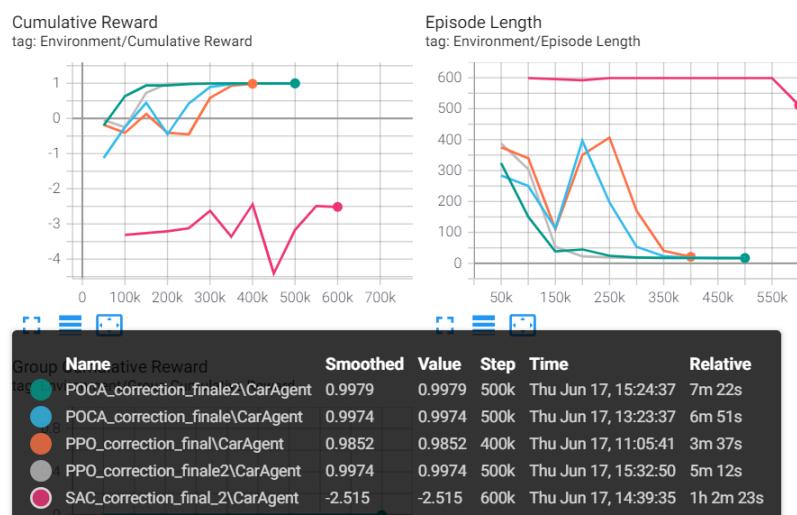


FIGURE 31 – Comparaison de l'évolution des courbes de récompense, du temps d'apprentissage, et du temps d'un épisode au cours du temps pour les algorithmes d'AR PPO, SAC et POCA.

Sur le graphique précédent figure 37, il peut être constaté que l'algorithme PPO possède un temps de convergence correct pour l'apprentissage de la politique du scénario simple. De plus, il paraît bien stable. Les performances de l'algorithme développé par l'équipe ML-Agents paraissent sensiblement équivalentes, puisqu'avec celui-ci, un peu moins de steps sont nécessaires avant de converger, mais le temps de calcul est légèrement supérieur. Pour l'algorithme SAC, après 1h d'entraînement, son apprentissage a été arrêté car cette méthode paraissait beaucoup trop longue pour un scénario aussi simple. Par la suite, l'utilisation de l'algorithme PPO a donc été conservée et le tuning des hyperparamètres a donc été réalisé avec celui-ci.

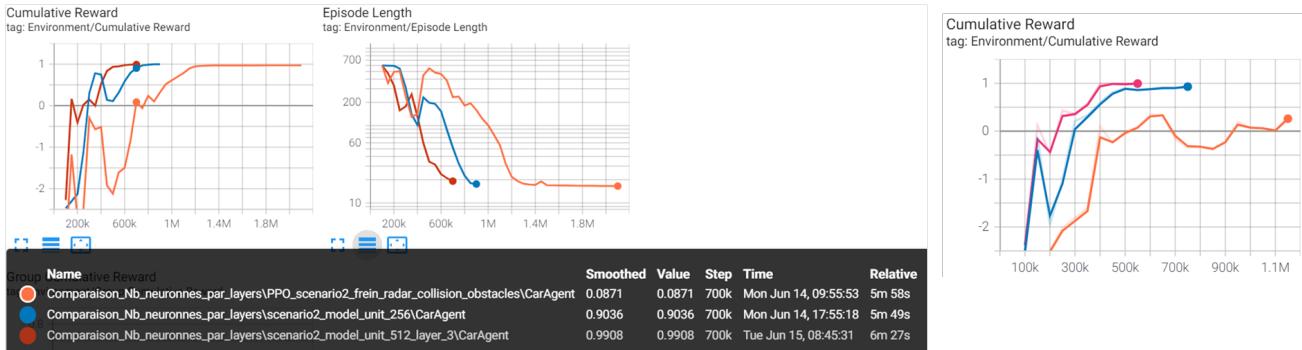


FIGURE 32 – Comparaison de l'évolution des courbes de récompense et du temps d'apprentissage pour différentes architectures de réseau de neurones approximant la politique : nombre de neurones par couche à gauche et nombre de couches à droite.

D'après la figure 32 à gauche, il semble que l'augmentation du nombre de neurones par couches permet d'accélérer l'apprentissage et donc de diminuer le temps de convergence. Les courbes orange, bleu et rouge, sont respectivement celles d'un apprentissage avec 128, 256 et 512 neurones pour un nombre de couches égal. La courbe rouge montre qu'avec plus de neurones, l'apprentissage paraît plus stable, mais prend aussi légèrement plus de temps de calcul. Sur la figure précédente, le graphique à droite possède les courbes orange, bleu et rose qui ont respectivement 1, 2 et 3 couches de neurones pour un nombre de neurones égal. De la même manière, on voit que l'ajout de couches accélère la convergence et stabilise l'apprentissage.



FIGURE 33 – Comparaison de l'entraînement du modèle d'AR en ajoutant la normalisation des observations, et en faisant varier la taille du `buffer` et le `time horizon` de l'algorithme PPO.

Sur la figure 33 précédente, d'autres hyperparamètres sont testés : tout à gauche, la courbe grise représente l'apprentissage de l'agent par défaut, la courbe bleu lorsque la pénalité à chaque instant ("existential reward") est enlevée, celle en rouge lorsque la normalisation des observations est ajoutée. Le graphique du milieu représente l'apprentissage de l'agent avec des tailles de `buffer` de 409600 pour la courbe en bleu, et de 4096 pour la courbe en rose. Le graphique de droite représente l'apprentissage de l'agent avec des tailles de `time horizon` (qui correspond au nombre de pas avant d'ajouter une expérience au buffer d'expériences), de 32 pour la courbe bleu et de 2048 pour la courbe rouge. Ces figures permettent donc de montrer que l'existential reward n'est pas nécessaire et que les observations doivent être normalisées. Les tailles de `buffer` et `time horizon` doivent être diminuées, de façon à collecter les expériences et mettre jour de manière plus fréquente le modèle et l'optimisation de la politique.

À l'issue de ces tests, le fichier `.yaml` a été paramétré comme dans l'annexe 15.

Finalement, pour le scénario simple de test, le graphique de gauche figure 34 suivante, montre que, le temps d'entraînement avant d'arriver à une convergence, a été divisé par environ 5 grâce au tuning des hyperparamètres. Il est représenté par les courbes bleue et rouge, par rapport à la configuration par défaut, représentée par la courbe de récompense rose. Additionnellement le Curriculum Learning [33] a été testé sur le scénario 1 pour essayer d'accélérer l'entraînement avec les recommandations de la documentation du plugin. Cela consiste à démarrer l'entraînement avec les positions basiques du scénario simple, pour l'agent et la place de parking. Par la suite, l'entraînement est arrêté puis redémarré en éloignant un peu plus l'agent de son objectif, et en repartant du modèle entraîné précédemment. Le résultat est obtenu au milieu de la figure 34 suivante.

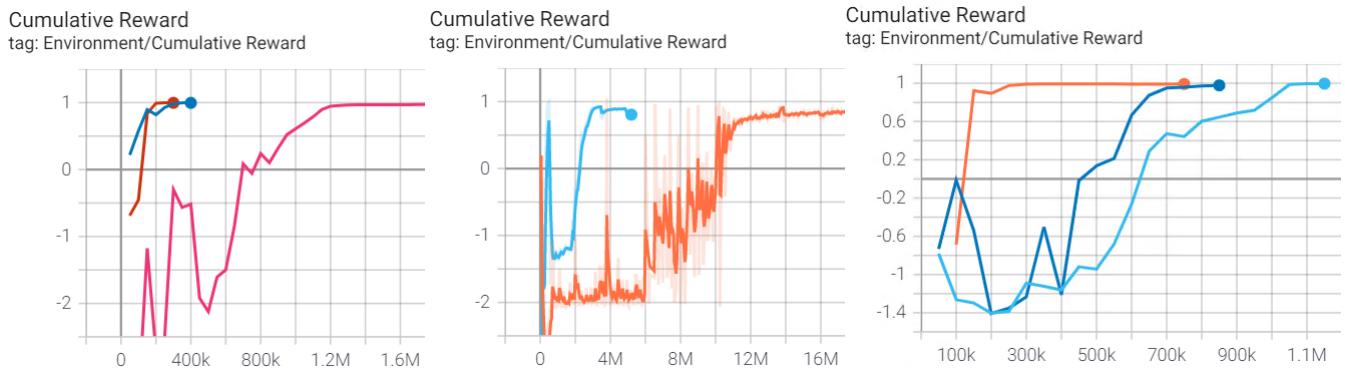


FIGURE 34 – Comparaison de l'entraînement du modèle d'AR avant et après tuning des hyperparamètres sur le modèle simple, avec et sans Curriculum Learning sur le scénario 1, puis en utilisant les modules de Curiosity, d'Imitation learning et de Behavior Cloning

Il semble que le Curriculum Learning permet d'accélérer l'apparition de la convergence, courbe bleu, par rapport à un entraînement classique, courbe orange. La "cassure" au sein de la courbe bleue est produite par l'arrêt de l'entraînement et l'augmentation de la difficulté de la tâche.

Pour finir, la dernière figure à droite, clôture la session de test sur scénario simple, par l'utilisation de 3 autres optimisations d'apprentissage supplémentaires fournies par le plugin. Les courbes bleu clair, bleu foncé et orange représente un entraînement sur le scénario simple avec respectivement l'ajout des optimisations "Curiosity", "Curiosity + GAIL", puis "Curiosity + GAIL + Behavior Cloning". Dans ML-Agent des paramètres permettent de pondérer l'utilisation de ces méthodes dans le comportement de l'agent [43]. Ces techniques sont décrites brièvement de manière à ne pas surcharger le rapport :

- Curiosity : A été créé pour les environnements à récompenses éparses [42], i.e. où les récompenses sont rares au cours d'un épisode. Ce module, schématisé en annexe 15, se base sur l'erreur de prédiction de l'agent : plus il est surpris, plus la récompense sera importante. Il utilise deux réseaux de neurones. Le premier "*inverse model*", permet de prédire l'action à effectuer et apprend à encoder l'observation actuelle et la suivante. Le second "*forward model*" prédit l'observation encodée suivante. Au final, ce module retourne une récompense intrinsèque à l'agent (*intrinsic reward*, indépendante de l'environnement). Son utilisation permet d'encourager l'agent à explorer l'espace d'états et lui permet de moins stagner en début d'entraînement.
- Generative Adversarial Imitation Learning (GAIL) : Le principe est d'utiliser un ensemble de démonstrations d'un expert, qui va jouer en quelque sorte le rôle d'adversaire de l'agent, et que ce dernier va vouloir dépasser. L'idée est de minimiser la différence entre les vrais expériences et les expériences générées par l'agent. Ce système se base sur les travaux du Generative Adversarial Network (GAN) [45] : Un module "*Discriminator*" distingue si une paire (observation, action) provient de la démonstration ou de l'agent, et un module "*Générateur*" apprend à générer des datas plausibles. La récompense est basée sur combien est-ce que la paire (observation,action) est proche de celle de la démonstration. La politique est donc extraite d'un ensemble de données avec le GAN et une fonction de coût convexe. Cette dernière minimise la divergence de Jensen-Shannon, annexe 45, dans le but de rapprocher la distribution de la politique de l'agent de celle de l'expert. Ce module est habituellement combiné avec Curiosity.

- Behavior Cloning : Ce principe permet d'entraîner l'agent à imiter exactement les actions disponibles dans une démonstration experte enregistrée 15. Les paires états-actions, (s_i^*, a_i^*) , sont traitées comme indépendantes et identiquement distribuées. L'agent entraîne sa politique π_θ en utilisant l'apprentissage supervisé et en minimisant la fonction de coût $L(a^*, \pi_\theta(s))$. Le Behavior Cloning fonctionne mieux lorsqu'il est utilisé en association avec le Gail, et lorsqu'il existe une démonstration pour presque tous les états possibles. L'idée est de mapper le comportement de l'agent apprenant avec celui des paires état-actions de la démonstration. Dans Unity, le Behavior Cloning est une "Pre-Training Option", qui agit seulement au début de l'entraînement (150k premiers steps), avec une certaine pondération par rapport à l'apprentissage par PPO (typiquement 0,5).

Remarques : Certaines limites aux deux derniers points existent. La démonstration experte enregistrée n'est pas parfaite : Elle correspond à environ 10 minutes d'expériences de l'agent contrôlé au clavier réalisant la tâche de stationnement (environ 10000 steps). Le Behavior Cloning suppose les paires (état, action) i.i.d., or l'AR se base sur un MDP, ce qui veut dire que le choix d'effectuer une action dans un certain état, implique l'état suivant. Cela casse donc l'hypothèse. De plus, pour le dernier point, si la démonstration de l'expert contient une erreur, ou s'il n'y a pas de données pour une certaine situation, l'agent va échouer (mais apprendra quand même au final avec l'algorithme PPO). Pour finir, ces 3 optimisations sont plus efficaces si elles sont utilisées ensemble, image de droite figure 34. Elles permettent de limiter le temps d'entraînement de l'agent, en trouvant plus vite quels états mènent à une récompense, au lieu de chercher longtemps en début d'épisode avant de recevoir un signal de récompense.

Après avoir réalisé ces tests et le tuning des hyperparamètres, le modèle doit être adapté au **scénario 2**, qui correspond au Use Case exact de KARlab, plus complexe. Il est décrit de la manière suivante :

"L'agent démarre sa tâche en partant de la zone bleue, à droite figure 29, avec une position et une orientation aléatoire . De même, la place de parking se trouve à une position aléatoire dans la zone rouge de la même figure. L'agent doit se garer avec alignement en moins d'une minute. L'action de freinage est utilisée et le stationnement autonome n'est validé seulement dans le cas où l'agent reste plus d'1 seconde à une distance de moins de 0.5 mètre de la place de parking. Lors de la présence d'un obstacle ou d'un piéton, le kart doit l'éviter, s'arrêter ou ralentir. La distance maximale de l'objectif est de 70 mètres".

Le modèle développé précédemment ne marchant plus ou n'étant plus optimal pour ce dernier scénario plus complexe, la fonction de récompense a été changée. Après de nombreuses tentatives, d'essais et de recherches, une pénalité proportionnelle à la distance entre l'agent et l'objectif a été ajoutée à chaque instant.

La **configuration finale** est donc la suivante :

- Espace d'états : L'angle de cap de l'agent ; sa position ; ses composantes de vitesses dans le plan ; la position de la place de parking ; Les composantes 3D de la distance restante entre la place de parking et l'agent ; Toutes les observations sont normalisées ; **Les 9 Raycasts décrits précédemment modélisent les observations radar.**
- Fonction de récompense : -1 si l'agent dépasse le temps de parking imparti (environ 1 minute) ; $\frac{-1}{\text{MaxSteps}}$ à chaque instant si l'agent choisit une action de torque négative ou d'angle des roues supérieur à $\pm 15^\circ$; $\frac{-1}{\text{MaxSteps}} * (\text{distanceToTarget})$ à chaque instant ; -1 si l'agent a une erreur d'alignement avec la place de parking de $\pm 10^\circ$ lorsqu'il est garé ; $+1$ si la distance entre l'agent et la place de parking est inférieure à 0.5 mètres durant plus de 1 seconde.

En utilisant cette configuration avec plusieurs étapes de Curriculum Learning, la courbe de récompense moyenne espérée obtenue est la suivante, image de gauche figure 35 suivante. Elle a été obtenue après environ 1 jours d'entraînement. L'image de droite correspond au même entraînement, mais en continuant l'entraînement de l'agent à partir du modèle entraîné, et avec l'ajout de 2 autres karts auxquels on a associé 2 autres places de parking et la politique obtenue précédemment. La courbe converge alors logiquement vers des performances moindres, car l'agent a plus de chance de rentrer en collision et d'échouer.

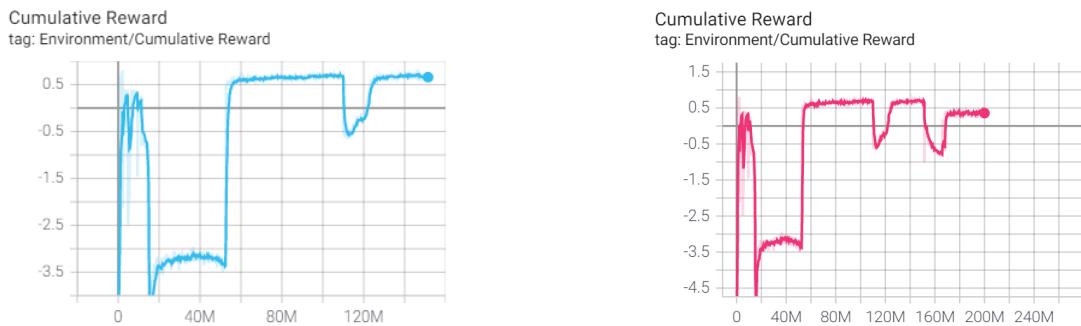


FIGURE 35 – Graphique de la courbe de récompense de l'agent lors de l'entraînement pour le scénario 2. Des étapes de Curriculum Learning sont effectuées. L'image de droite est obtenue lorsque l'entraînement est repris en ajoutant 2 autres karts

Cette configuration permet d'obtenir un premier modèle s'approchant d'une politique optimale pour le stationnement autonome : Lors du test du modèle, le kart obtient un taux de succès de stationnement de 90% dont 97% correspondent à un alignement avec le parking. Les cas où l'agent échoue correspondent à des situations ambigus, où le piéton fait un angle mort avec l'agent, ou quand le véhicule commence l'épisode devant plusieurs blocs. La fin de l'entraînement de ce scénario est illustré avec la [VIDEO 4], et l'inférence est disponible avec la [VIDEO 5].

Note : N'ayant pas réussi à apprendre à l'agent le fait de gérer sa vitesse, le `maxMotorTorque` du véhicule à été volontairement abaissé, de manière à borner la vitesse du kart à environ 13 km/h. De plus, lors d'un entraînement, quand la récompense espérée moyenne se stabilise, la pénalité proportionnelle à la distance à l'objectif est retirée, pour voir si la courbe d'apprentissage converge vers 1, la récompense maximale possible.

La dernière session d'entraînement vise à obtenir le résultat obtenu précédemment, mais de manière direct et plus rapide.

Il peut être constaté que sans Curriculum Learning, et en entraînant directement l'agent, le système n'apprend pas (figure 36, courbe orange, image de gauche). Les optimisations décrites précédemment sont alors ajoutées. Il y a convergence vers une politique optimale, pour l'ajout de "Curiosity", "Curiosity + GAIL", puis "Curiosity + GAIL + Behavior Cloning", après respectivement 11h, 6h et 4h. Ces courbes sont disponible sur la figure 36 suivante, en bleu, gris et rose, toujours dans le même ordre. Le "saut" présent à la fin de l'entraînement sur ces courbes de récompenses correspond à la **Note** précédente.

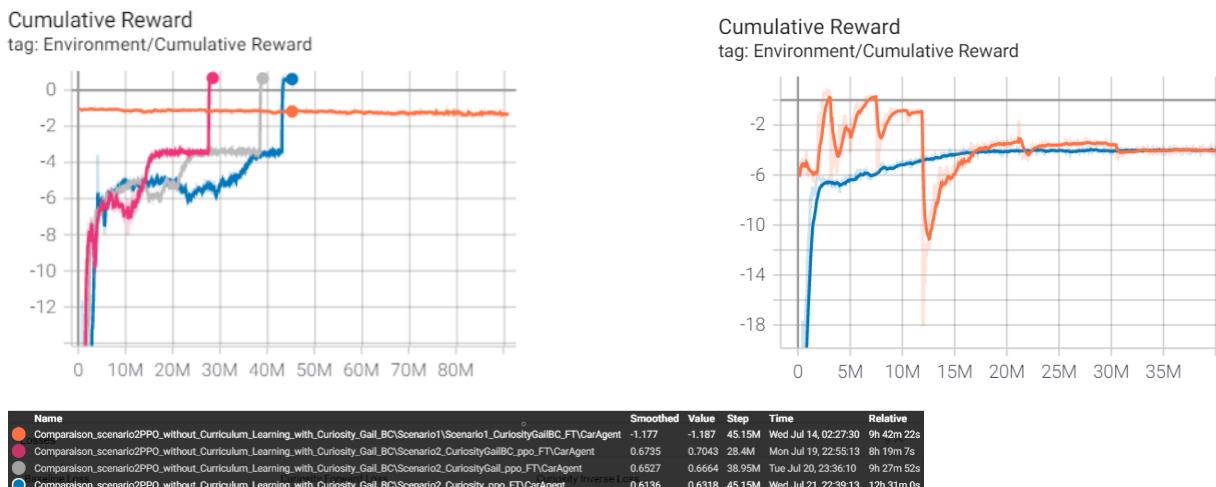


FIGURE 36 – Courbes de récompense de l'agent lors de l'entraînement pour le scénario 2, gauche. À droite, celles d'un entraînement de l'agent avec et sans Curriculum Learning sur le scénario 2 avec 2 autres karts.

Le principe est aussi réitéré avec 2 karts supplémentaires dans le scénario 2. Pour cela, l'épisode consiste à faire apparaître aléatoirement dans la zone en bleu, deux autres agents suivant la politique obtenue précédemment. Chacun d'eux possède une place de parking associée, apparaissant aléatoirement dans la zone rouge. L'illustration de la modélisation Unity peut être retrouvée en annexe 15. L'entraînement est alors lancé uniquement pour un kart, qui correspond à l'agent apprenant le comportement optimal au scénario 2, tout en gérant la présence d'autres kart. De plus, le même entraînement a été testé avec Curriculum Learning, en divisant la tâche en 6 étapes de difficultés (agent et parking proche, puis plus éloignés, puis ajoutant les obstacles, etc). Enfin, pour chacune de ces étapes, une démonstration a été enregistrée (durant 10 minutes pour environ 50 épisodes et 6000 steps), de manière à appliquer une optimisation maximale pour chaque étape de Curriculum Learning. L'entraînement peut être retrouvé [VIDEO 6]. Les courbes de récompenses se trouvent figure 36 à droite. La courbe en orange de cette figure représente la récompense moyenne espérée de l'apprentissage **avec Curriculum Learning** et avec les 3 optimisations, pour le scénario 2, auquel 2 karts sont ajoutés. La courbe en bleu représente la récompense moyenne espérée de l'apprentissage avec les mêmes conditions d'entraînement mais pour apprentissage "from scratch" **sans Curriculum Learning**.

11.1.4 Résultats

Les résultats finaux de ces entraînements sont présents sur la figure 37 suivante. Le graphique de gauche correspond aux valeurs de récompenses moyennes espérées pour la fin des entraînements détaillés précédemment (présents figure 36 en orange et bleu). La courbe rouge, elle, correspond à la même configuration que la courbe bleue, mais en utilisant l'algorithme POCA. L'image de droite de la figure ci-dessous représente un résumé des différents entraînements et résultats obtenus pour les différents scénarios.



FIGURE 37 – Comparaisons et résultats finaux des modèles d'AR développés pour le stationnement autonome du kart.

D'après les figures précédentes (figure 36 gauche, et figure 37), les résultats et conclusions pour les modèles d'AR développés pour le du Use Case 3 de KARlab sont les suivants :

- L'entraînement de l'agent, en utilisant l'algorithme PPO, auquel on combine les optimisations de Curiosity, GAIL et Behavior Cloning, et en utilisant le Curriculum Learning, courbe orange figure 37 précédente, obtient un taux de réussite légèrement meilleur que sans l'utilisation de Curriculum learning, courbe bleu : 75.8% contre 73.7%. Néanmoins, le meilleur résultat met plus de temps à converger, et il est plus contraignant, car une démonstration doit être enregistrée pour chaque étape de Curriculum Learning.

- Lorsque l'agent réussit son stationnement autonome, il est presque tout le temps aligné avec la place de parking, avec un taux d'alignement d'environ 98%, lorsque le kart réussit à se garer (386/393, figure 37).
- Le tableau de droite de la figure 37 précédente, montre en bas à droite que lorsque l'agent échoue, c'est en majeure partie à cause d'une collision avec un autre kart. Cela paraît normal, car ce sont les obstacles qui se déplacent le plus rapidement et donc les plus difficiles à éviter. Des collisions avec un piéton ou d'autres obstacles statiques existent aussi de temps à autre.
- L'entraînement en utilisant l'algorithme POCA, auquel on combine les optimisation de Curiosity, GAIL et Behavior Cloning, sans utiliser le Curriculum Learning, courbe rouge figure 37 précédente, paraît obtenir des résultats légèrement meilleurs que la configuration orange, en un temps moindre : 77.3% de réussite pour environ 5h30. Bien que testé, il ne dispose d'aucune documentation (qui doit être publiée dans une prochaine release du plugin).

Remarque : Étant assez chronophage, certains de ces entraînements ont été effectués la nuit, ou sans contrôler leurs avancements, durant plusieurs heures. Les temps de convergence indiqués à droite du tableau ont donc été approximés, en soustrayant les temps d'entraînement où la courbe de récompense stagnait.

Finalement, pour la résolution finale du scénario 2, auquel 2 karts sont ajoutés, le modèle indiqué précédemment en tant que "**configuration finale**" semble le plus approprié. L'optimisation de la politique par l'algorithme PPO, tuné comme précédemment, sans Curriculum Learning, et auquel est ajouté les 3 optimisations Curiosity, GAIL et Behavior Cloning, apparaît comme une solution avec le meilleur compromis (courbe bleue figures précédentes). De cette manière, une politique adaptée au Use Case 3 de KARlab peut être obtenue en une durée de temps raisonnable et sans trop d'efforts ou de contraintes (en laissant l'apprentissage tourner durant une nuit par exemple).

Le modèle développé permet donc de rendre assez stable un Apprentissage par Renforcement Profond destiné au kart. Le tuning des hyperparamètres et l'utilisation de méthodes pour optimiser l'entraînement, ont un réel impact, et sont très utiles pour obtenir un résultat satisfaisant dans un temps correct. Un résumé schématique du système solution à la problématique étudiée est fourni figure 38 suivante : Le bloc du milieu "*KARlab model Use Case 3*", correspond au modèle entraîné, approximant une politique optimale pour le stationnement autonome du kart. Ce réseau de neurones est simplifié et schématisé en annexe 15 et le fichier résultant au format .onnx est détaillé en annexe 15.

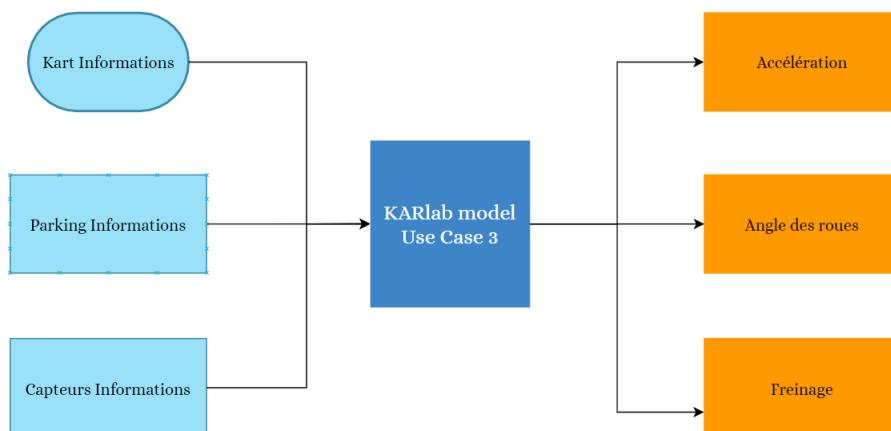


FIGURE 38 – Entrées et sorties du système permettant le stationnement autonome du kart

À l'issue de cette première PoC proposée avec Unity, un premier avancement de la mise en place de ce modèle d'AR a été réalisé sur le logiciel CARLA Simulator.

11.2 Modélisation du stationnement autonome par AR sur Carla simulator

CARLA est un simulateur open source conçu pour la recherche et le développement en matière de conduite autonome, en servant d'outil facilement accessible et personnalisable par les utilisateurs [32]. Il a été développé par le Computer Vision Center et l'Université de Barcelone. Ce projet est sponsorisé par des grandes entreprises telles que Intel, Toyota ou encore General Motors. Le simulateur permet de représenter un monde virtuel et d'y incorporer de nombreux acteurs (véhicules, piétons) dans un monde prédéfini (milieu urbain, autoroute, circuit). Le simulateur est basé sur le moteur graphique Unreal Engine 4, et utilise la norme OpenDRIVE pour définir les routes et les environnements urbains. Le contrôle de la simulation est assuré par une API gérée en Python et en C++, qui s'enrichit constamment au fur et à mesure que le projet avance.

11.2.1 Le simulateur

CARLA possède une architecture de serveur multi-client : Plusieurs clients peuvent se connecter au même serveur de test pour pouvoir contrôler plusieurs véhicules et piétons, aussi appelés "*acteurs*". Le serveur est responsable de tout ce qui concerne la simulation elle-même : le rendu des capteurs, le calcul de la physique, la mise à jour de l'état du monde et de ses acteurs. Le côté client est constitué d'une somme de modules clients, contrôle la logique des acteurs en scène et fixe les conditions du monde. L'interface proposée permet aux utilisateurs de développer des scripts en Python pour contrôler ces acteurs.

CARLA permet l'utilisation de différents capteurs utiles à la conduite autonome, tels que le Radar, le LIDAR, de multiples caméras, un capteur de profondeur, le GPS ou encore l'IMU. Ils peuvent être inclus à un même véhicule et diffusent des informations sur leur environnement. Ces informations peuvent être récupérées et stockées.

Un des points forts de ce simulateur est qu'il possède une documentation complète, et une bonne communauté : Il existe assez de forum où les créateurs et développeurs de CARLA échangent sur le logiciel et sur l'implémentation des différentes fonctionnalités possibles. Par ailleurs, il est possible de créer ses propres carte de simulation et ses propres véhicules, pour les utiliser ensuite dans CARLA. Cela nécessite néanmoins de passer par une modélisation avec l'éditeur Unreal Engine.

11.2.2 Mise en place du modèle d'AR sur CARLA

Modélisation sur CARLA

N'ayant trouvé aucune source ou aucun projet réalisé se rapprochant du stationnement autonome étudié, l'environnement d'AR pour le stationnement autonome du kart est d'abord recréé sur CARLA, à l'image de celui créé sur Unity. Comme illustré en annexe 15, la modélisation suivante est faite : L'environnement de parking est chargé avec la carte de simulation "Town05" fournit par CARLA ; La position d'une place de parking est relevée (Log correspondant au rond vert sur les images figures 51 et 52) ; Le véhicule est modélisée en prenant le véhicule "BMW Isetta" de CARLA Simulator, et ses caractéristiques physiques (poids et taille des roues) sont celles du kart de KARlab ; Le radar a été positionné à l'avant du véhicule, et 10 mesures radar sont relevées toutes les 0.1 seconde : Chaque point permet d'avoir la distance de l'obstacle détecté (profondeur) ainsi que son angle dans le plan azimutal. Dans l'annexe, figures 51 et 52, ces points sont représentés par des carrés blancs.

Note : Ne pouvant pas mieux modéliser le kart du projet ou changer sa "Bounding Box" sur CARLA, c'est le véhicule mentionné ci-dessus qui a été choisi pour la modélisation.

Une fois les observations normalisées et récupérées (figure 50 annexe) de la même manière que pour le modèle développé sur Unity, l'environnement CARLA est alors prêt pour que l'agent apprenne à réaliser un stationnement autonome par AR. La figure suivante 39 schématisé comment l'AR se modélise à travers CARLA.

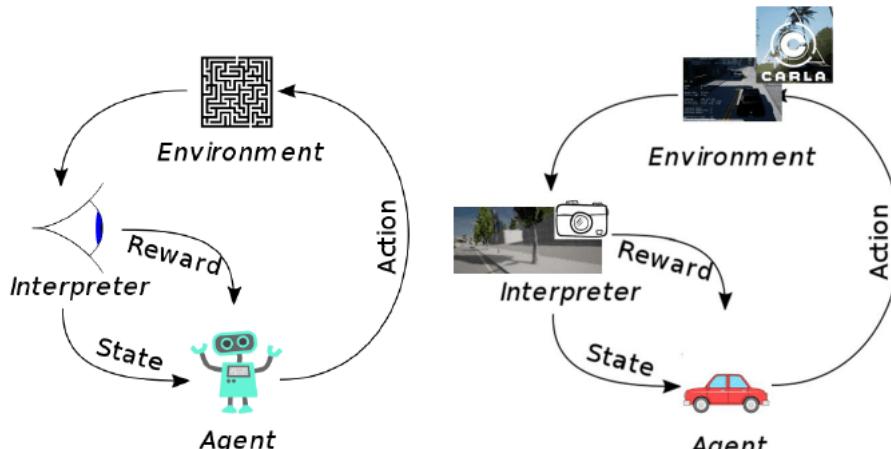


Figure 1 R Learning

Figure 2 R Learning in Carla

FIGURE 39 – Schéma de la modélisation du modèle d'apprentissage par renforcement avec CARLA (droite)

TF-Agents

Après de nombreuses recherches sur l'utilisation d'un AR avec CARLA, le framework TF-Agents [49] de TensorFlow a été choisi pour l'apprentissage. Cette bibliothèque facilite la conception, la mise en œuvre et le test d'algorithmes d'apprentissage par renforcement en fournissant des composants modulaires, modifiables et extensibles. Cette bibliothèque permet une itération rapide du code. Dans TF-Agents, les environnements peuvent être implémentés en Python ou en TensorFlow. Les environnements Python sont généralement plus faciles à implémenter, à comprendre et à déboguer. Le flux de travail le plus courant consiste à implémenter un environnement en Python et à ensuite utiliser un *wrapper* pour le convertir automatiquement en TensorFlow. Un wrapper d'environnement prend un environnement Python et renvoie une version modifiée de l'environnement. La conversion des environnements Python en "TFEnvs" permet à tensorflow de paralléliser les opérations. Des objets tenseurs sont alors générés au lieu de tableaux. La figure 40 suivante résume l'utilisation classique de ce framework :

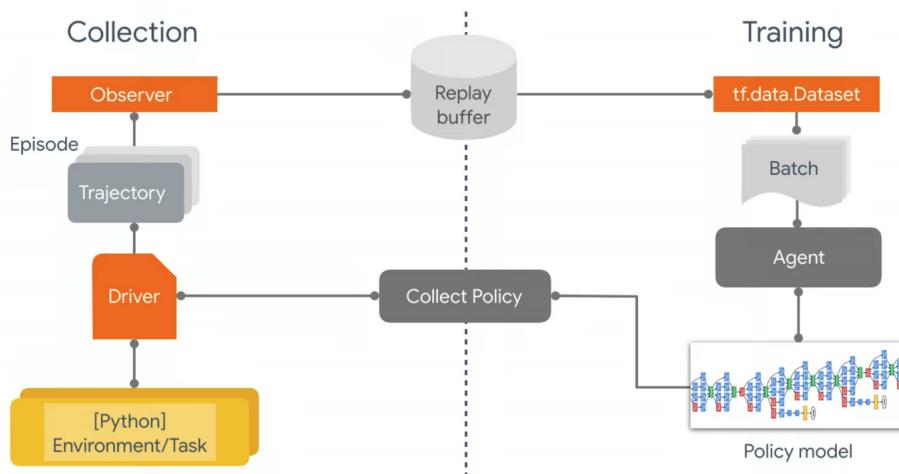


FIGURE 40 – Architecture et organisation de l'utilisation de TF-Agents pour l'AR

Fonctionnement

Les environnements Python de TF-Agents ont une méthode "`step(action)`", qui applique une action à l'environnement et retourne en échange : l'"observation" de l'agent, le "reward" (la récompense), le "discount" (la pondération de la récompense, le " γ ") et le "step type" de l'état suivant (si c'est le premier pas de temps, le dernier, ou bien un pas de temps intermédiaire).

Cet environnement fournit également une méthode "`reset()`" qui démarre un nouvel épisode et fournit un "`TimeStep`" initial (Analogue à "`OnEpisodeBegin()`" dans ML-Agents). De plus, dans le cas présent de création de classe pour l'environnement d'AR en Python avant son wrapping, une fonction "`init()`" doit aussi être complétée en spécifiant les types des valeurs de l'espace d'états et d'actions. Pour ce projet elles ont toutes été paramétrées en "`float32`".

L'avancement du Use Case 3 sur CARLA est arrêté au point ci-dessus : Un premier test a été fait pour l'itération d'un seul step avec le framework, mais a rencontré une erreur car les `observations` et `time steps` générés ne suivent pas les formes ou les types corrects tels que définis dans les spécifications de l'espace d'états ou d'actions.

12 Retour sur expérience

Prise de recul sur le travail effectué

Lors de ce stage, l'état de l'art de l'AR a été long mais nécessaire, étant vraiment un domaine particulier du Machine Learning. Après avoir constaté que peu de sources et d'applications concrètes de l'AR existent dans un cas de stationnement autonome, le plugin Unity ML-Agents a été choisi pour modéliser le problème, et faire une première PoC fonctionnelle. Cette décision peut être justifiée : Elle semble avoir permis un grand gain de temps, puisque grâce au plugin ML-Agents, plus de 100 simulations d'AR ont été réalisées. Bien que Unity demeure tout de même un simulateur physique 3D, les résultats obtenus paraissent satisfaisants. Le modèle entraîné permet au kart d'effectuer un stationnement autonome en s'alignant avec la place de parking. De plus, il peut ralentir, s'arrêter et éviter les obstacles ou piétons (avec tout de même quelques collisions et limites, bien sûr.). Un deuxième choix technique a aussi été réalisé en commençant à mettre en place le framework TF-Agents dans le but de remodéliser l'environnement de parking sur CARLA simulator, et réaliser un AR optimisé grâce à cette librairie de TensorFlow. Une fois pris en main et assimilé, ce framework paraît assez modulaire et répondre aux besoins technologiques du modèle.

L'équipe KARlab évolue en méthodologie AGILE en sprints de 3 semaines. Les tâches que chacun doit effectuer sont donc "*sizées*". Pour ma part, le temps passé pour les effectuer correspondait souvent à cette durée prévue, excepté pour les tâches sur lesquelles l'équipe avait moins de visibilité (étendue de l'état de l'art de l'AR, ou le choix de la technologie pour réaliser un AR avec CARLA Simulator). De plus, le résultat final de ce stage est une première PoC du Use Case 3 sur Unity et une mise en place de la même PoC sur CARLA Simulator. Cet avancement se rapproche donc d'une solution de la problématique (à savoir que le fichier ".onnx" obtenu grâce à Unity peut potentiellement être directement embarqué sur le kart réel). Une partie conséquente de l'AR et de sa modélisation possible a aussi été documentée pour la suite.

Impressions et difficultés rencontrés

Bien sûr, il va sans dire que ce stage et sa réalisation furent semés d'embûches. Tout d'abord, le sujet sur lequel j'ai travaillé est assez complexe, et avoir une expertise du domaine de l'AR nécessiterait beaucoup plus que 6 mois. Je suis aussi arrivé sur le Use Case 3 de KARlab pour le stationnement autonome du kart, sans aucune connaissance, et sans disposer de documentation ou travail préalable interne. Les spécifications de cette fonctionnalité restaient ouvertes, et nous devions prendre nous-même, mon maître de stage et moi, les décisions techniques. Ma première PoC a été réalisée avec un langage et un éditeur que je n'avais jamais utilisés auparavant, à savoir respectivement le C# et Unity. Bien que modulaire grâce au plugin ML-Agents, l'entraînement par AR d'un agent prend du temps et est aussi très chronophage. Je n'avais jamais utilisé le simulateur CARLA, et le développement des scripts en Python restait quelque chose de lointain. Pour finir, certains problèmes de proxy, internes à l'entreprise, pouvait ralentir l'exécution de certaines tâches et ont du être résolus.

J'ai néanmoins beaucoup aimé la méthode AGILE, qui permet à mon sens d'être productif, et de prendre du recul sur le travail fourni au quotidien. La grande autonomie dont j'ai bénéficié m'a permise de développer mon analyse critique de papiers scientifiques, ma réflexion sur la conception de modèles de machine learning, et mon autonomie en programmation, et en résolution de problèmes propres à partir d'un mix de sources. J'ai enfin pu profiter d'une très bonne cohésion d'équipe.

Activités finalement réalisées durant le stage
- Intégration dans le projet de l'entreprise
- Etat de l'art de l'Apprentissage par Renforcement
- Etat de l'art de cette technique appliquée au véhicule et stationnement autonome
- Montées en compétences en C#, Unity, CARLA simulator et Python
- Proposition et validation de modèles d'AR pour le stationnement autonome
- Tests et expérimentations avec plusieurs modèles de stationnement autonome
- Evaluations des performances et études d'optimisations supplémentaires
- Présentations, communications et documentations des travaux

TABLE 2 – Réalisations durant mon stage sur le projet KARlab

13 Conclusion

Ce projet m'a permis d'évoluer et de développer une grande autonomie dans la modélisation, le développement et la résolution pratique d'un problème de Machine Learning. J'ai pu répondre en partie à une problématique novatrice sur laquelle les solutions sont en plein développement. Ce stage de 6 mois m'a fait découvrir le monde du travail grâce au projet de recherches et développements KARlab. J'ai évolué dans une équipe pluridisciplinaire, et pu voir passer une très grande variété de tâches effectuées, propres à chacun (systèmes embarqués, réseaux, réalité mixte, développements web etc). L'apprentissage par renforcement et son application sont des thématiques très intéressantes mais peuvent aussi se révéler très complexes. L'importante documentation réalisée pendant le stage m'a permis en quelque sorte d'entrevoir le monde de la recherche, tout en travaillant dans une équipe projet, en entreprise et en méthode AGILE. L'utilisation de technologies et la pratique m'ont fait bénéficier d'une montée en compétences sur des langages orientés objets et différents simulateurs ou logiciels de modélisation 3D. J'ai découvert le travail en équipe sur un projet avancé de grande envergure, et eu l'opportunité de présenter à maintes reprises mon travail. Pour la suite, ce stage m'a conforté dans l'idée de continuer dans la spécialisation d'ingénierie de la filière Télécommunications - I2SC.

Finalement, grâce à l'approximation d'une politique par réseau de neurones, sur simulateur, le kart peut effectuer un stationnement autonome à partir de la perception de ses informations physiques et de celles du radar. À chaque instant, il peut prendre la décision optimale pour la réalisation de sa tâche. Le rendu visuel est disponible [VIDEO 7]. Le modèle résultant se trouve dans un format universel .onnx, qui permet potentiellement de s'en ressourcer avec différentes technologies. La mise en place du modèle d'AR pour le stationnement autonome sur CARLA permet d'approcher une seconde PoC sur un simulateur plus complet.

Des travaux futurs à partir de mon stage sont de développer le modèle initié sur CARLA, pour entraîner un agent de la même manière que ce qui a été fait sur Unity, ou bien alors de se servir du modèle obtenu .onnx déjà entraîné. Pour finir, il est important de bien penser à la manière dont la solution sera embarquée sur le kart réel, et la façon dont le modèle sera affiné sur un parking réel.

14 Bibliographie

Références

- [1] Dibangoye, J. S. (2020). OpenClassrooms. *Concevez un algorithme d'apprentissage par renforcement profond.* <https://openclassrooms.com/en/courses/5098641-utilisez-lapprentissage-par-renforcement-avec-un-drone/>
- [2] Choudhary, A. (2020, 24 mai). *Nuts Bolts of Reinforcement Learning : Model Based Planning using Dynamic Programming.* Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2018/09/reinforcement-learning-model-based-planning-dynamic-programming/>
- [3] Gérard, P. (2007–2008). *Apprentissage par Renforcement, Apprentissage Numérique.* Université de Paris 13 - LIPN Master MICR. <https://lipn.univ-paris13.fr/~gerard/docs/cours/ar-cours-support.pdf>
- [4] Professor S.Mahadevan University of Massachusetts at Amherst. (2016). *RL Lecture 5 : Monte Carlo Methods.* <http://www-edlab.cs.umass.edu/>. <http://www-edlab.cs.umass.edu/cs689/lectures/RL%20Lecture%205.pdf>
- [5] Silver D., Google DeepMind. (2020). *Lecture 7 : Policy Gradient.* <https://www.davidsilver.uk/wp-content/uploads/2020/03/pg.pdf>
- [6] Richard S. Sutton and Andrew G. Barto. (2015). *Reinforcement Learning : An Introduction.* <https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf>
- [7] Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K. (2016). *Asynchronous Methods for Deep Reinforcement Learning.* <https://arxiv.org/pdf/1602.01783.pdf>
- [8] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. (2017). *Proximal Policy Optimization Algorithms.* <https://arxiv.org/pdf/1707.06347.pdf>
- [9] Brownlee, J. (2019, 25 octobre). *Difference Between a Batch and an Epoch in a Neural Network. Machine Learning Mastery.* <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/>
- [10] Schulman, J., Levine, S., Abbeel, P., Jordan, M.I., Moritz, P. (2015). *Trust Region Policy Optimization* <https://arxiv.org/pdf/1502.05477.pdf>
- [11] Wikipedia contributors. (2021b, juillet 1). *Divergence de Kullback-Leibler.* https://fr.wikipedia.org/wiki/Divergence_de_Kullback-Leibler
- [12] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, et al. (ICML, Jun 2014, Beijing, China). *Deterministic policy gradient algorithms* <https://hal.inria.fr/hal-00938992/document>
- [13] Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D. (2016). *Continuous control with deep reinforcement learning.* CoRR, abs/1509.02971. <https://arxiv.org/pdf/1509.02971.pdf>
- [14] Wang, L., Wang, K., Pan, C., Xu, W., Aslam, N. (2020). *Joint Trajectory and Passive Beamforming Design for Intelligent Reflecting Surface-Aided UAV Communications : A Deep Reinforcement Learning Approach..* <https://arxiv.org/pdf/2007.08380.pdf>
- [15] Haarnoja, T., Zhou, A., Abbeel, P., Levine, S. (2018). *Soft Actor-Critic : Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.* ICML. <https://arxiv.org/pdf/1801.01290.pdf>
- [16] Weng, L. (2018, 8 avril). *Policy Gradient Algorithms.* <https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html>
- [17] Spinning Up, OpenAI (2018). *Soft Actor-Critic — Spinning Up documentation.* <https://spinningup.openai.com/en/latest/algorithms/sac.html>

- [18] Rădulescu, R., Mannion, P., Roijers, D.M. et al. Autonomous Agents and Multi-Agent Systems 34, 10 (2020). *Multi-objective multi-agent decision making : a utility-based analysis and survey.* <https://doi.org/10.1007/s10458-019-09433-x>
- [19] Choudhary A., Analytics Vidhya (2020, 24 mai). *Reinforcement Learning Guide : Solving the Multi-Armed Bandit Problem from Scratch in Python.* <https://www.analyticsvidhya.com/blog/2018/09/reinforcement-multi-armed-bandit-scratch-python/> 9.06560.pdf
- [20] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D. (30 Jan 2019). *Deep Reinforcement Learning that Matters.* <https://arxiv.org/pdf/1709.06560.pdf>
- [21] A. Folkers, M. Rick and C. Büskens (2019). *Controlling an Autonomous Vehicle with Deep Reinforcement Learning.* IEEE Intelligent Vehicles Symposium (IV), Paris, France, 2019, pp. 2025-2031, doi : 10.1109/IVS.2019.8814124. <https://arxiv.org/pdf/1909.12153.pdf>
- [22] L. Sommer, M. Rick, A. Folkers, and C. Buskens (2018). *AO-Car : Transfer of Space Technology to Autonomous Driving with the use of WORHP.* 7th International Conference on Astrodynamics Tools and Techniques (ICATT).
- [23] Zhuang, Y., Gu, Q., Wang, B., Luo, J., Zhang, H., Liu, W. (2018). *Robust Auto-parking : Reinforcement Learning based Real-time Planning Approach with Domain Template.* <https://openreview.net/pdf?id=BylpU61C97>
- [24] Zhang, P., Xiong, L., Yu, Z., Fang, P., Yan, S., Yao, J., Zhou, Y. (2019). *Reinforcement Learning-Based End-to-End Parking for Automatic Parking System.* Sensors, 19(18), 3996. <https://doi.org/10.3390/s19183996>
- [25] Zhang, J., Chen, H., Song, S., Hu, F. (2020). *Reinforcement Learning-Based Motion Planning for Automatic Parking System.* IEEE Access, 8, 154485-154501. <https://doi.org/10.1109/access.2020.3017770>
- [26] Zhang, J., Chen, H., Song, S., Hu, F. (2020). *Trajectory Planning for Automated Parking Systems Using Deep Reinforcement Learning* Int.J Automot. Technol. 21, 881-887 <https://doi.org/10.1007/s12239-020-0085-9>
- [27] B. Thunyapoo, C. Ratchadakorntham, P. Siricharoen and W. Susutti. (2020). *Self-Parking Car Simulation using Reinforcement Learning Approach for Moderate Complexity Parking Scenario* 17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), Phuket, Thailand, pp. 576-579. doi : 10.1109/ECTI-CON49241.2020.9158298
- [28] Fehér, R., Aradi, S., Hegedűs, F., Bécsi, T., Gáspár, P. (2019). *Hybrid DDPG Approach for Vehicle Motion Planning.* Proceedings of the 16th International Conference on Informatics in Control, Automation and Robotics, 422. <https://doi.org/10.5220/0007955504220429>
- [29] Feher, A., Aradi, S., Becsi, T., Gaspar, P., Szalay, Z. (2020). *Proving Ground Test of a DDPG-based Vehicle Trajectory Planner.* European Control Conference (ECC). doi :10.23919/ecc51009.2020.9143675
- [30] E. Bejar and A. Morán (2019) *Reverse Parking a Car-Like Mobile Robot with Deep Reinforcement Learning and Preview Control.* IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2019, pp. 0377-0383, doi : 10.1109/CCWC.2019.8666613.
- [31] Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M., Lange, D. (2018). *Unity : A General Platform for Intelligent Agents* ArXiv, abs/1809.02627. <https://arxiv.org/pdf/1809.02627.pdf>
- [32] Dosovitskiy, A., Ros, G., Codevilla, F., López, A.M., & Koltun, V. (2017). *CARLA : An Open Urban Driving Simulator.* ArXiv, abs/1711.03938. <https://arxiv.org/pdf/1711.03938.pdf>
- [33] Bengio, Y., Louradour, J., Collobert, R., Weston, J. (2009). *Curriculum learning.* Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09. Published. <https://doi.org/10.1145/1553374.1553380>
- [34] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2018). *Deep Reinforcement Learning that Matters.* AAAI. <https://arxiv.org/pdf/1709.06560.pdf>

- [35] Duan, Y., Chen, X., Houthooft, R., Schulman, J., & Abbeel, P. (2016). *Benchmarking Deep Reinforcement Learning for Continuous Control*. ICML. <https://arxiv.org/pdf/1604.06778.pdf>
- [36] Unity Technologies (2020). *Unity - Manual : Wheel Collider*. <https://docs.unity3d.com/Manual/class-WheelCollider.html>
- [37] Vallecillos D. (2020). *Various examples of ml-agents while teaching concepts about it in YouTube*. GitHub - dilmerv/UnityMLEssentials <https://github.com/dilmerv/UnityMLEssentials> et <https://www.youtube.com/watch?v=IcatCC9Rikk>
- [38] Unity-Technologies (2020b). *Unity ML-Agents Toolkit* GitHub - Unity-Technologies/ml-agents <https://github.com/Unity-Technologies/ml-agents>
- [39] Unity-Technologies (2020b). *Making a New Learning Environment* GitHub - Unity-Technologies/ml-agents https://github.com/Unity-Technologies/ml-agents/blob/release_17_docs/docs/Learning-Environment-Create-New.md
- [40] Bai, Junjie and Lu, Fang and Zhang, Ke and others (2019). *ONNX : Open Neural Network Exchange* GitHub <https://github.com/onnx/onnx>
- [41] Google (2015). *Citing TensorFlow* <https://www.tensorflow.org/about/bib?hl=en> et <https://github.com/tensorflow/tensorflow>
- [42] Pathak, D., Agrawal, P., Efros, A.A., Darrell, T. (2017). *Curiosity-Driven Exploration by Self-Supervised Prediction*. 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 488-489. <https://arxiv.org/pdf/1705.05363.pdf>
- [43] Juliani, A. (2018, 26 juin). *Solving sparse-reward tasks with Curiosity*. Unity Blog. <https://blog.unity.com/technology/solving-sparse-reward-tasks-with-curiosity>
- [44] Ho, J., Ermon, S. (2016). *Generative Adversarial Imitation Learning*. NIPS. <https://papers.nips.cc/paper/2016/file/cc7e2b878868cbae992d1fb743995d8f-Paper.pdf>
- [45] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A.C., Bengio, Y. (2014). *Generative Adversarial Networks*. NIPS. <https://arxiv.org/pdf/1406.2661.pdf>
- [46] Goecks, V.G., Gremillion, G., Lawhern, V., Valasek, J., Waytowich, N.R. (2019). *Integrating Behavior Cloning and Reinforcement Learning for Improved Performance in Dense and Sparse Reward Environments* <https://arxiv.org/pdf/1910.04281.pdf>
- [47] Ai, S. (2019, 20 septembre). *A brief overview of Imitation Learning* SmartLab AI. Medium. <https://smartlabai.medium.com/a-brief-overview-of-imitation-learning-8a8a75c44a9c>
- [48] Ai, S. (2019, 20 septembre). *A brief overview of Imitation Learning* SmartLab AI. Medium. <https://smartlabai.medium.com/a-brief-overview-of-imitation-learning-8a8a75c44a9c>
- [49] Sergio Guadarrama and Anoop Korattikara and Oscar Ramirez and Pablo Castro and Ethan Holly and Sam Fishman and Ke Wang and Ekaterina Gonina and Neal Wu and Efi Kokiopoulou and Luciano Sbaiz and Jamie Smith and Gábor Bartók and Jesse Berent and Chris Harris and Vincent Vanhoucke and Eugene Brevdo (2018). *TF-Agents : A library for Reinforcement Learning in TensorFlow* <https://github.com/tensorflow/agents>; <https://www.youtube.com/watch?v=-TTziY7EmUA>; <https://youtu.be/U7g7-Jzj9qo>; https://www.tensorflow.org/agents/tutorials/2_environments_tutorial#wrapping_a_python_environment_in_tensorflow

15 Annexes

Principes d'un réseau de neurones artificiel

Un réseau de neurones permet l'approximation non-linéaire d'une fonction. C'est un apprentissage dit "profond" qui, en connaissance d'un objectif, va apprendre une représentation nécessaire pour l'approximer et proposer une solution à partir de données brutes. Un réseau de neurones possède une ou plusieurs entrées et sorties. C'est au final une composition de plusieurs fonctions paramétriques : Pour un réseau classique "feedforward", chaque couche possède plusieurs neurones, dont chacune représente une transformation affine. Sur chaque neurone est appliquée une fonction d'activation. Les couches sont ensuite interconnectées entre elles comme sur la figure 48.

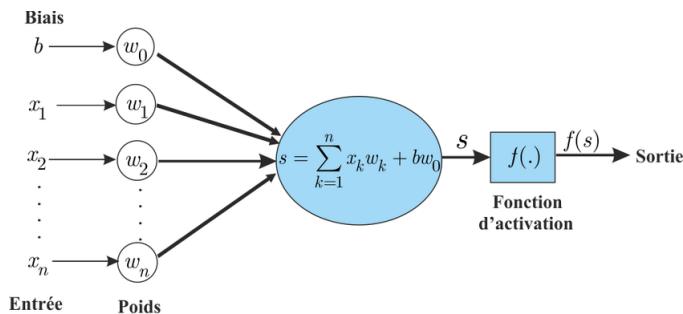


FIGURE 41 – Principes de fonctionnement d'un neurone artificiel

Principes d'une descente de gradient

La première étape de l'entraînement d'un réseau de neurones, pour satisfaire un objectif précis, consiste à calculer la différence entre la sortie réelle et la sortie désirée grâce à une fonction de perte (Erreur Quadratique Moyenne, Log Vraisemblance etc.). L'erreur va alors être rétro-propagée dans l'ordre inverse du réseau en calculant le gradient de la fonction de perte associée : Les dérivées partielles de chaque composante par rapport aux autres composantes sont calculées ; En multipliant toutes ces dérivées partielles les unes avec les autres, cela nous aide à calculer la dérivée partielle de la fonction de perte en fonction de toutes les composantes de cette fonction composée, et en particulier des paramètres eux-mêmes. Ces derniers sont alors ajustés dans la meilleure direction afin d'optimiser la fonction de perte. L'apprentissage des réseaux de neurones profonds repose sur le partage des poids. L'identification et l'exploitation des régularités conduit à des interdépendances entre paramètres.

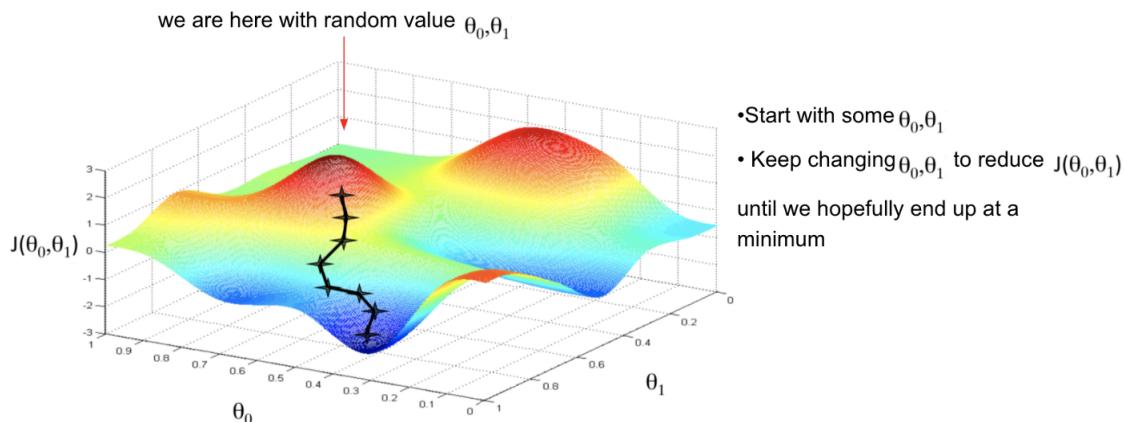


FIGURE 42 – Schéma d'une descente de gradient minimisant une fonction de coût par rapport à deux vecteurs θ_0 et θ_1

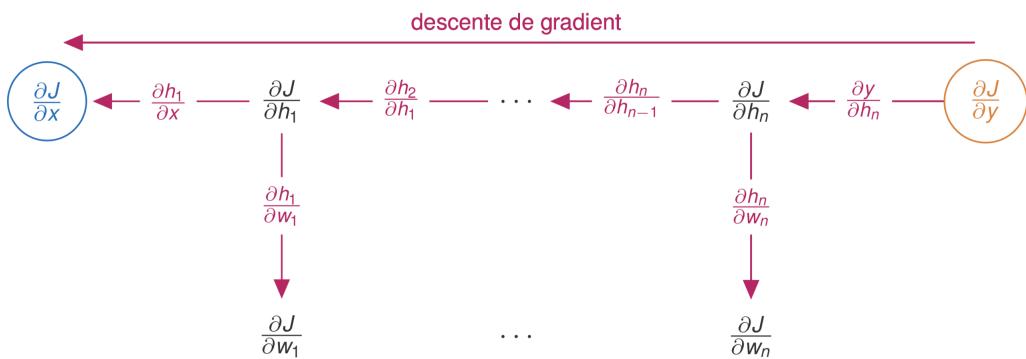


FIGURE 43 – Schéma d'une descente de gradient en calculant les dérivées partielles.

Fichier de configuration .YAML par défaut :

```

1 behaviors:
2   CarAgent:
3     trainer_type: ppo
4     hyperparameters:
5       batch_size: 10
6       buffer_size: 100
7       learning_rate: 3.0e-4
8       beta: 5.0e-4
9       epsilon: 0.2
10      lambd: 0.99
11      num_epoch: 3
12      learning_rate_schedule: linear
13    network_settings:
14      normalize: false
15      hidden_units: 128
16      num_layers: 2
17    reward_signals:
18      extrinsic:
19        gamma: 0.99
20        strength: 1.0
21      max_steps: 500000
22      time_horizon: 64
23      summary_freq: 10000

```

Fichier de configuration .YAML final :

```

1 behaviors:
2   CarAgent:
3     trainer_type: ppo
4     hyperparameters:
5       batch_size: 1024
6       buffer_size: 4096
7       learning_rate: 1.0e-4
8       beta: 1.0e-4
9       epsilon: 0.2
10      lambd: 0.95
11      num_epoch: 3
12      learning_rate_schedule: linear
13    network_settings:
14      normalize: true
15      hidden_units: 512
16      num_layers: 3
17    reward_signals:
18      extrinsic:
19        gamma: 0.99
20        strength: 1.0
21      max_steps: 200000000
22      time_horizon: 32
23      summary_freq: 50000

```

Curiosity

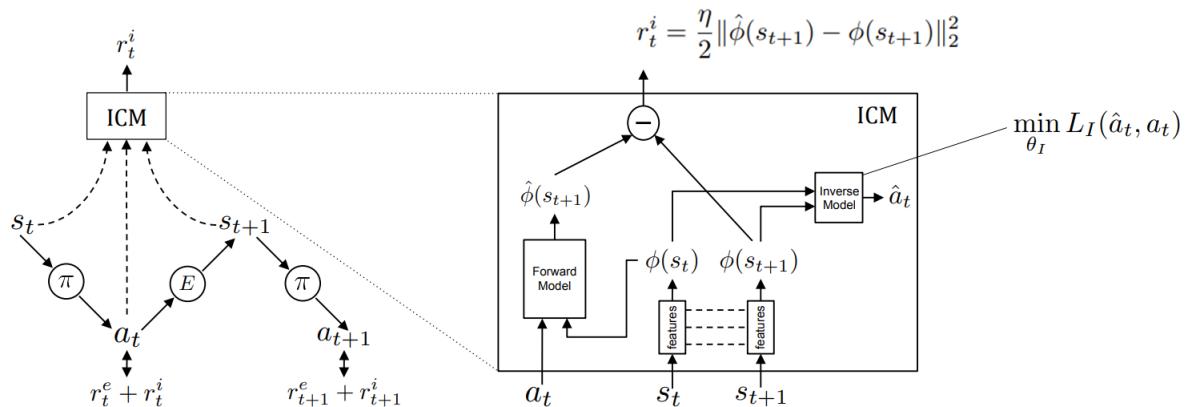


FIGURE 44 – Schéma du module de Curiosity

Generative Adversariale Imitation Learning

Algorithm 1 Generative adversarial imitation learning

- 1: **Input:** Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
- 2: **for** $i = 0, 1, 2, \dots$ **do**
- 3: Sample trajectories $\tau_i \sim \pi_{\theta_i}$
- 4: Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i} [\nabla_w \log(D_w(s, a))] + \hat{\mathbb{E}}_{\tau_E} [\nabla_w \log(1 - D_w(s, a))] \quad (17)$$

- 5: Take a policy step from θ_i to θ_{i+1} , using the TRPO rule with cost function $\log(D_{w_{i+1}}(s, a))$. Specifically, take a KL-constrained natural gradient step with

$$\hat{\mathbb{E}}_{\tau_i} [\nabla_\theta \log \pi_\theta(a|s) Q(s, a)] - \lambda \nabla_\theta H(\pi_\theta), \quad (18)$$

where $Q(\bar{s}, \bar{a}) = \hat{\mathbb{E}}_{\tau_i} [\log(D_{w_{i+1}}(s, a)) | s_0 = \bar{s}, a_0 = \bar{a}]$

- 6: **end for**
-

$$\underset{\pi}{\text{minimize}} \psi_{\text{GA}}^*(\rho_\pi - \rho_{\pi_E}) - \lambda H(\pi) = D_{\text{JS}}(\rho_\pi, \rho_{\pi_E}) - \lambda H(\pi)$$

FIGURE 45 – Pseudocode et fonction de coût du Generative adversarial imitation learning

Behavior Cloning

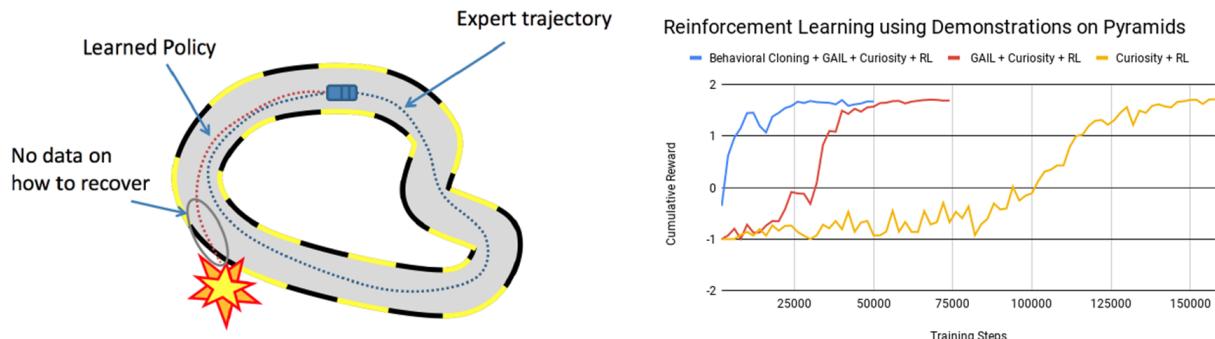


FIGURE 46 – Schéma du Behavior Cloning et documentation ML-Agent sur les modules précédent

Illustration du scénario 2 en ajoutant 2 karts et détections de l'agent

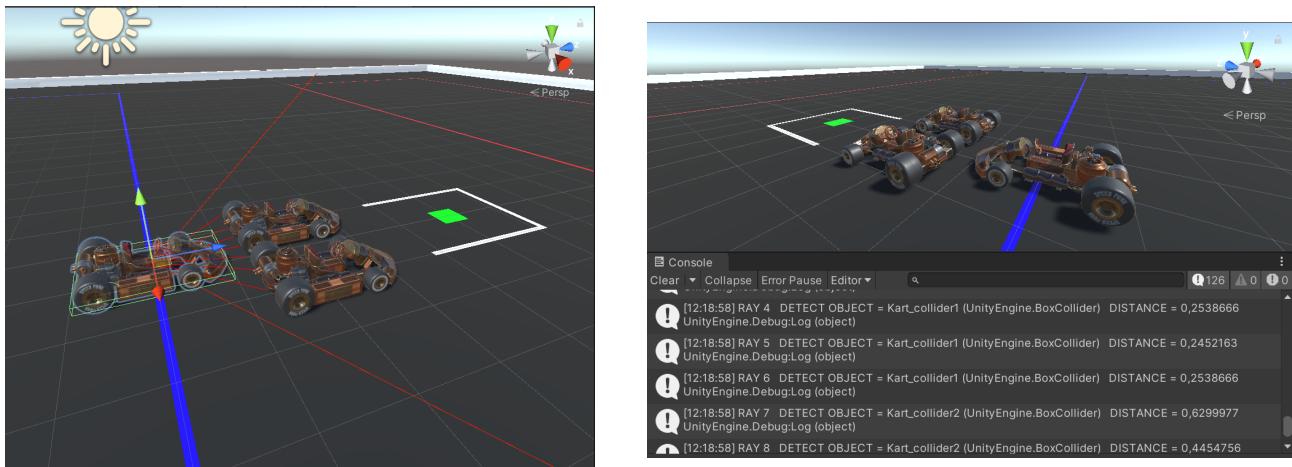


FIGURE 47 – Modélisation Unity du scénario 2 en ajoutant 2 autres karts et impacts des Raycast de l'agent sur ces derniers, à gauche. L'image de droite représente l'affichage des détections de l'agent lorsque la scène est lancée.

Schéma simplifié du réseau de neurones obtenu pour le stationnement autonome

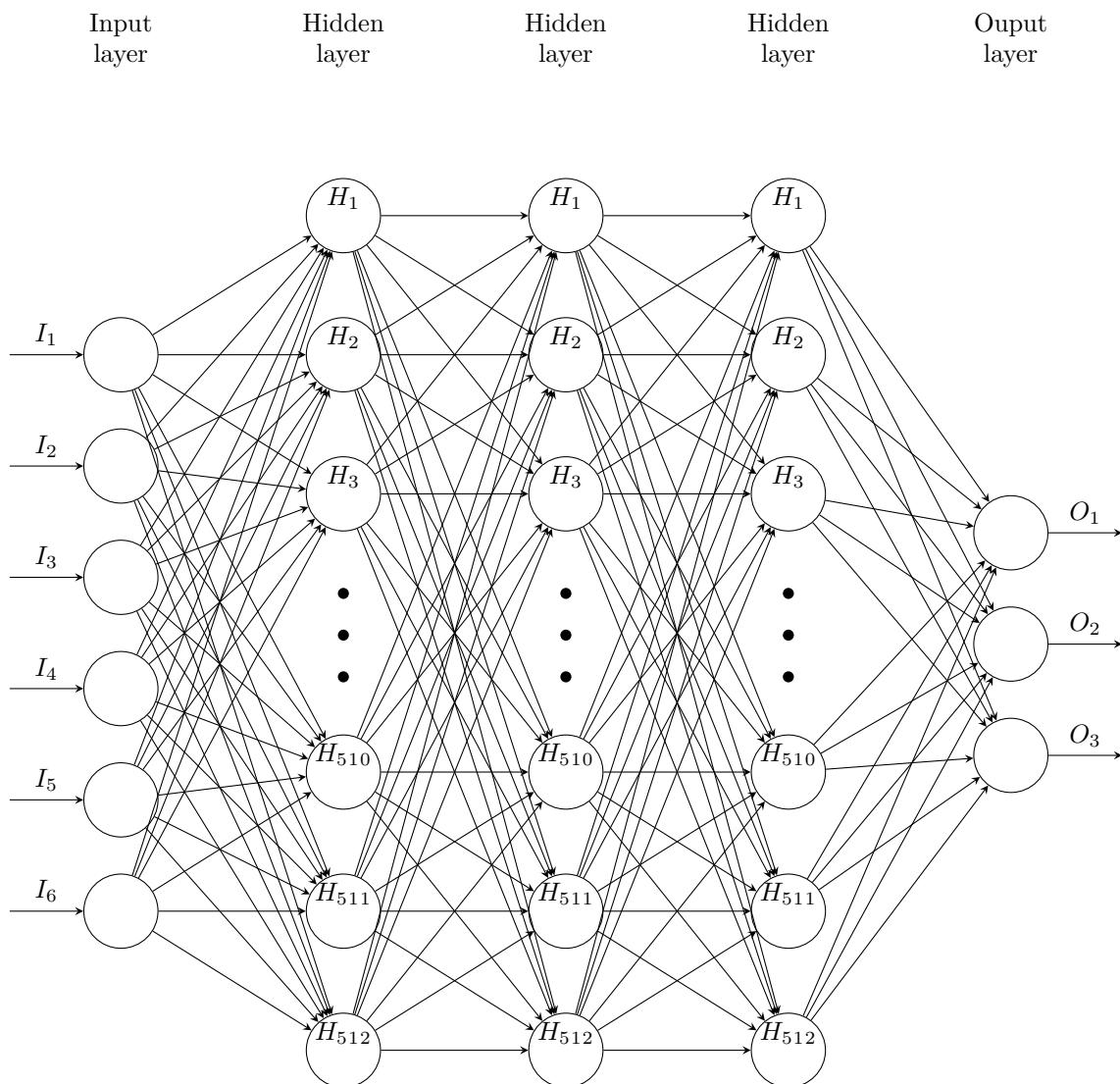


FIGURE 48 – Schéma simplifiée de l'architecture du réseau de neurones solution au Use Case 3 de stationnement autonome. Les entrées I_i sont respectivement la position du véhicule, celle de la place de parking associée, l'angle de cap du véhicule, ses deux composantes de vitesse dans le plan, les composantes en 3D de la distance restante entre l'agent et l'objectif, et l'information du radar ; Les "Hidden Layers" correspondent aux trois couches de 512 neurones chacunes ; Les sorties O_i sont respectivement l'accélération l'angle des roues et le freinage.

Architecture finale du modèle .onnx obtenu

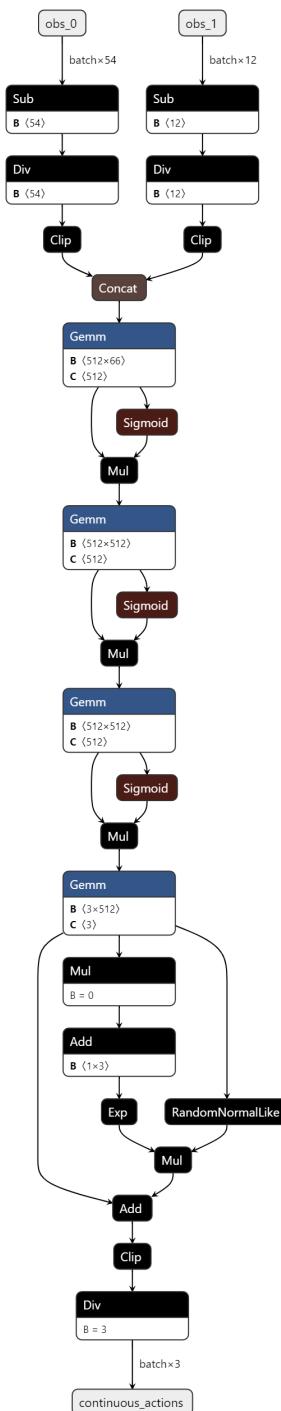


FIGURE 49 – Architecture du modèle .onnx final obtenu pour le stationnement autonome du kart.

Illustration de l'avancement du Use Case 3 mis en place sur CARLA Simulator

```

Vehicule Infos:

- Vehicule Position: Location(x=14.000000, y=-24.400000, z=0.200000)
- Vehicule Yaw: -179.99996948242188
- Vehicule Velocity: Vector3D(x=0.000000, y=0.000000, z=0.000000)
- Parking Target location: Vector3D(x=10.300000, y=-24.400000, z=0.100000)
- Remaining distance Parking-Vehicule: Vector3D(x=-3.700000, y=0.000000, z=-0.100000)

OBSERVATIONS:

- Vehicle position norm: Vector3D(x=0.497657, y=-0.867345, z=0.007109)
- Vehicle yaw norm: -0.49999991522894965
- Vehicle velocity: Vector3D(x=-nan(ind), y=-nan(ind), z=-nan(ind))
- Parking location norm: Vector3D(x=0.388898, y=-0.921273, z=0.003776)
- Remaining distance norm: Vector3D(x=-0.999635, y=0.000000, z=-0.027017)

```

FIGURE 50 – Logs de la récupération des informations et des observations normalisées du véhicule, nécessaires au modèle d'AR.



FIGURE 51 – Environnement de parking pour le stationnement autonome sur CARLA : Il s'y trouve le parking et les places pour se garer, le véhicule (l'agent) représentant le kart en bleu au milieu, le marquage de la place de parking (rond vert) et celui de la position du radar (croix rouge), ainsi que les détections radar (carrés blancs).

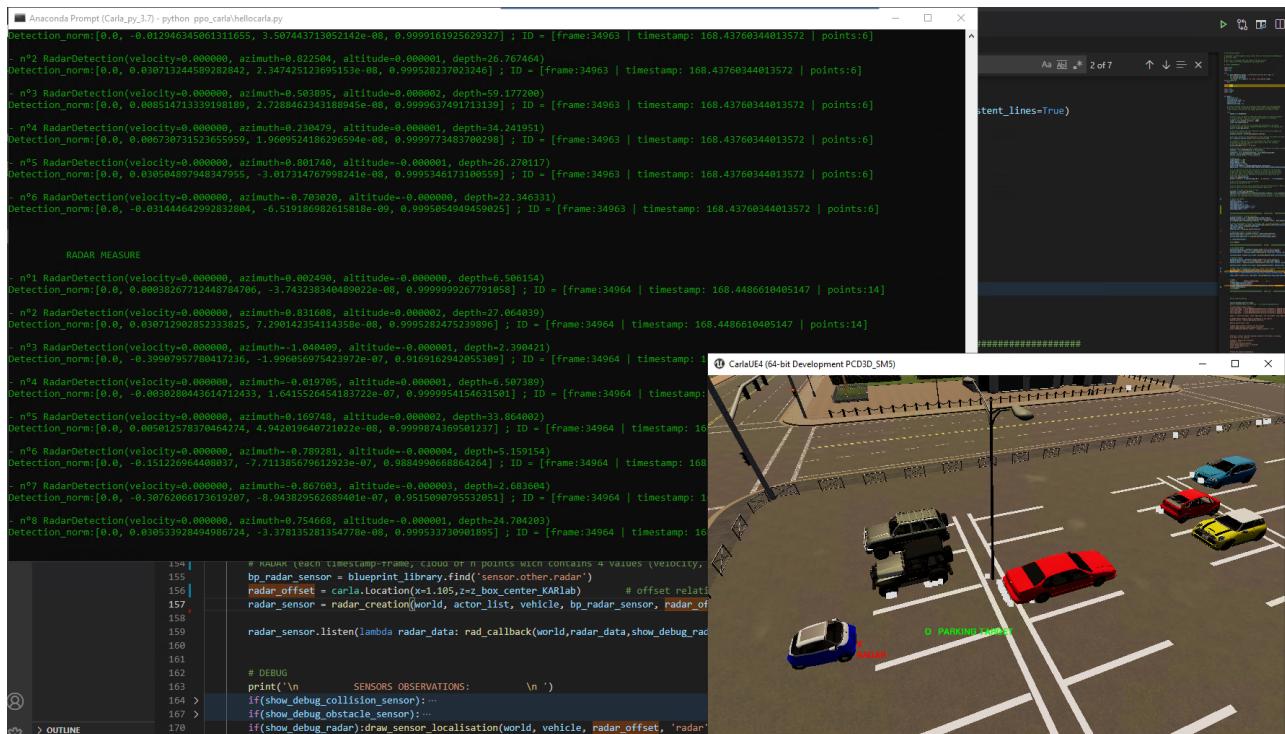


FIGURE 52 – Capture d'écran contenant une autre vue de l'environnement de parking pour le stationnement autonome sur CARLA illustré figure 51. La capture d'écran comporte aussi le détail de la récupération des détections radar effectuées toutes les 0.1 seconde, en vert à gauche.