Apprentissage par renforcement du contrôle d'un véhicule autonome à partir de la vision

Marin Toromanoff

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à MINES ParisTech

Apprentissage par renforcement du contrôle d'un véhicule autonome à partir de la vision

Soutenue par
Marin TOROMANOFF
Le 31 mars 2021

École doctorale n°621
ISMME Ingénierie des Systèmes, Matériaux, Mécanique, Energétique

Spécialité
Informatique temps réel, robotique et automatique

Composition du jury :

Pierre-Yves OUDEYER DR., Inria Bordeaux	*Président*
Olivier PIETQUIN Prof., Université de Lille/GoogleBrain	*Rapporteur*
Thierry CHATEAU Prof., Université Clermont Auvergne	*Rapporteur*
Christian GAGNE Prof., Université Laval Québec	*Examinateur*
Rémi MUNOS DR., Université de Lille/DeepMind	*Examinateur*
Véronique CHERFAOUI Prof., Université de Compiègne	*Examinateur*
Fabien MOUTARDE Prof., Mines ParisTech/CAOR	*Directeur de thèse*
Émilie WIRBEL Dr., Nvidia	*Examinateur*

MINES ParisTech | PSL

Table des matières

1 Introduction **1**
- 1.1 Description générale des systèmes de conduite autonome . . . 2
- 1.2 Buts poursuivis par cette thèse . . . 3
  - 1.2.1 Objectif de la thèse . . . 3
  - 1.2.2 Comment apprendre les actions ? . . . 4
- 1.3 Cadre de la thèse . . . 6
- 1.4 Structure du manuscrit . . . 7

2 Apprentissage par renforcement et application au véhicule autonome **9**
- 2.1 Bases de l'apprentissage par renforcement . . . 9
- 2.2 Politique et fonction de valeur . . . 11
- 2.3 Les familles d'algorithmes d'apprentissage par renforcement . . . 12
  - 2.3.1 Fondés sur la fonction de valeur . . . 13
  - 2.3.2 Fondés sur la politique . . . 18
  - 2.3.3 Fondés sur un modèle de l'environnement . . . 21
- 2.4 Apprentissage par renforcement et réseau de neurones . . . 23
  - 2.4.1 Deep Q Network (DQN) : le "premier" algorithme d'apprentissage par renforcement profond . . . 23
  - 2.4.2 Asynchronous Advantage Actor-Critic (A3C) : le "premier" algorithme acteur-critique de DRL . . . 25
- 2.5 Comment appliquer du DRL à la voiture autonome ? . . . 26
  - 2.5.1 Choisir l'environnement : les simulateurs pour la voiture autonome 26
  - 2.5.2 Choisir la représentation de l'état et des actions : Apprentissage des capteurs au contrôle, ou représentation *haut niveau* . . . 33
  - 2.5.3 Quel algorithme de Renforcement choisir ? . . . 37
- 2.6 Conclusion : Choix techniques et d'application de la thèse . . . 39

3 Travail préliminaire : DRL sur un jeu de course réaliste **41**
- 3.1 Introduction . . . 42
  - 3.1.1 Travaux connexes : DRL pour les jeux de course . . . 44
- 3.2 Méthode . . . 44
  - 3.2.1 Apprentissage distribué sur plusieurs machines . . . 46
  - 3.2.2 Espace d'actions . . . 46
  - 3.2.3 Fonction de récompense . . . 47
- 3.3 Expériences . . . 48
- 3.4 Discussion : Qu'a apporté ce travail préliminaire à la thèse ? . . . 51
  - 3.4.1 Description des faiblesses de ce travail préliminaire . . . 51
  - 3.4.2 Les enseignements tirés de ce travail préliminaire . . . 52
- 3.5 Conclusion . . . 52

4 Nouvel algorithme état de l'art sur Atari : Rainbow-IQN Ape-X 55
4.1 Introduction . . . 56
4.2 Travaux connexes . . . 57
4.2.1 Reproductibilité et comparaison dans le DRL . . . 58
4.2.2 Apprentissage par renforcement profond fondé sur la fonction de valeur . . . 61
4.3 SABER : un benchmark Atari standardisé . . . 64
4.3.1 Temps maximal d'un épisode . . . 64
4.3.2 La référence des records du monde humains . . . 65
4.3.3 Description de SABER . . . 66
4.4 Rainbow-IQN Ape-X . . . 67
4.5 Expériences . . . 69
4.5.1 Infrastructure d'entraînement : Utilisation d'un centre de calcul à distance, le CCRT . . . 70
4.5.2 Évaluation de Rainbow avec SABER . . . 70
4.5.3 Évaluation de Rainbow-IQN avec SABER . . . 71
4.6 Conclusion . . . 72
4.6.1 Pourquoi le renforcement est-il si mauvais sur les jeux Atari ? . . . 73
4.6.2 Quid de la voiture autonome ? . . . 74

5 RL pour la conduite urbaine en utilisant des *indices implicites* 75
5.1 Introduction . . . 76
5.2 Travaux connexes . . . 78
5.2.1 Apprentissage de représentation pour le renforcement . . . 78
5.2.2 Application du DRL à la conduite urbaine de bout-en-bout . . . 82
5.3 Le benchmark CARLA et le challenge CARLA . . . 84
5.3.1 Le benchmark CARLA . . . 84
5.3.2 Le challenge CARLA . . . 85
5.4 Méthodes . . . 86
5.4.1 Apprentissage par renforcement fondé sur la fonction de valeur : Rainbow-IQN Ape-X . . . 87
5.4.2 Choix de la fonction de récompense . . . 87
5.4.3 Architecture de notre réseau de neurones . . . 88
5.5 Défis et solutions pour appliquer du RL à la conduite urbaine . . . 91
5.5.1 Entraîner par renforcement un agent ayant des entrées de haute dimension : *indices implicites* . . . 91
5.5.2 Gérer la discrétisation des actions . . . 93
5.6 Expériences et études d'ablations . . . 95
5.6.1 Introduction d'une situation de test et d'une métrique pour une évaluation commune . . . 95
5.6.2 Études d'ablations sur la phase d'entraînement supervisé de l'encodeur . . . 96
5.6.3 Études d'ablations sur la phase d'entraînement par renforcement . 98

5.6.4 Étude de la généralisation sur des villes jamais vues 99
5.6.5 Comparaison sur le benchmark CARLA 100
5.7 Conclusion 102

6 Intégration sur véhicule réel : Apprentissage par imitation 103
6.1 Introduction 104
6.1.1 Présentation de l'apprentissage par imitation 104
6.1.2 L'apprentissage par imitation appliqué à la voiture autonome . . . 108
6.2 Contrôle latéral sur véhicule réel : configuration expérimentale 115
6.2.1 Présentation des capteurs et des véhicules réels 115
6.2.2 Les cas d'utilisation ciblés et les base de données associées 116
6.2.3 Architecture de notre réseau de neurones 118
6.3 Méthode et contributions 119
6.3.1 Augmentation de point de vue et correction de l'angle au volant associé avec une seule caméra fisheye 119
6.3.2 Simulateur fondé sur des images réelles 121
6.3.3 Sélection de données 122
6.4 Expériences 124
6.4.1 Définition de scénarios d'évaluation pour le simulateur 124
6.4.2 Résultats des différentes sélections de données 125
6.4.3 Bagging de différents modèles 126
6.4.4 Résultat qualitatif sur Grand Theft Auto (GTA) 127
6.4.5 Résultats sur voiture réelle 127
6.5 Conclusion 127

7 Apprentissage par renforcement sur un véhicule réel 129
7.1 Introduction 129
7.2 Travaux connexes : DRL appliqué sur un véhicule réel 130
7.2.1 Learning to Drive in a Day : Première application sur véhicule réel 130
7.2.2 VISTA : Renforcement avec simulateur fondé sur des images réelles 130
7.3 Génération de point de vue avec estimation de profondeur 133
7.3.1 Estimation de profondeur à partir d'une seule caméra fisheye : FisheyeDistanceNet 134
7.3.2 Projection à partir d'une carte de profondeur et de l'image RGB . 139
7.4 Renforcement pour le contrôle latéral en environnement urbain 143
7.4.1 Apprentissage d'indices implicites pour le contrôle latéral 143
7.4.2 Apprentissage par renforcement sur simulateur fondé sur des images réelles 144
7.4.3 Résultat et comparaison du renforcement par rapport à l'imitation 145
7.5 Renforcement pour le contrôle longitudinal 147
7.5.1 Association temporelle ou spatiale pour le simulateur 149
7.5.2 Création d'une base de données pour les indices implicites 150
7.5.3 Apprentissage d'indices implicites pour le contrôle longitudinal . . 151
7.5.4 Configuration de l'apprentissage par renforcement 153

7.5.5 Résultat préliminaire de l'apprentissage par renforcement 154
7.6 Conclusion 155

8 Conclusion **157**
8.1 Contributions 157
8.2 Publications 158
8.3 Travaux futurs 159
8.4 Messages clés de cette thèse 160

Bibliographie **161**

Chapitre 1

Introduction

Dès 1994[1], les chercheurs du projet Prometheus ont fait une démonstration de conduite autonome sur autoroute avec leur véhicule VITA (pour *Vision Information Technology Application*). Durant les trois *DARPA Grand Challenge* de 2004 à 2007[2], plusieurs équipes de chercheurs ont réussi à naviguer sans conducteur humain sur plusieurs centaines de kilomètres dans le désert et dans des zones urbaines contrôlées. Ces performances impressionnantes ont été considérées comme une preuve que les voitures autonomes étaient réalisables et ont suscité un grand optimisme quant au fait que cette technologie deviendrait rapidement disponible. Aujourd'hui, dix ans après le lancement du projet de Google en 2009 pour développer une voiture sans conducteur, les premières flottes de robot-taxis permettent aux utilisateurs de se déplacer dans certaines zones géographiques spécifiques[3]. En parallèle aux projets visant à instaurer directement une voiture totalement autonome, les véhicules privés sont équipés avec un nombre croissant de systèmes d'assistance à la conduite (ADAS) comme la direction active pour le maintien de voie, le contrôle de la vitesse et le changement de voie sur autoroute ainsi que des systèmes permettant de se garer automatiquement. Cependant, même si ces systèmes d'aide à la conduite fonctionnent de mieux et mieux, le conducteur doit toujours rester attentif pour pouvoir intervenir à tout moment.

A la lumière de ces progrès, le déploiement à grande échelle de véhicules autonomes augure d'une nette amélioration de la sécurité sur les routes et d'une diminution des coûts des transports. Cependant, les experts restent encore incertains sur le temps qu'il reste avant d'avoir des voitures autonomes accessibles au plus grand nombre capable de véritablement conduire en environnement urbain.

Pour mieux comprendre les difficultés technologiques qui restent encore à résoudre, nous allons d'abord rapidement introduire l'architecture générale des systèmes de conduite autonome actuels en Section 1.1. Dans un deuxième temps, en Section 1.2, nous présenterons les buts poursuivis par cette thèse ainsi que les approches possibles pour atteindre ces objectifs. Finalement, nous décrirons en Section 1.4 la structure chapitre par chapitre de l'ensemble de ce manuscrit.

1. https://media.daimler.com/marsMediaSite/en/instance/ko/The-PROMETHEUS-project-launched-in-1986-Pioneering-autonomous-driving.xhtml?oid=13744534
2. https://www.darpa.mil/news-events/2014-03-13
3. https://venturebeat.com/2019/12/05/waymo-one-ios-app-launch/

1.1 Description générale des systèmes de conduite autonome

Les architectures des systèmes de conduite autonome sont généralement séparées en trois modules distincts illustrés sur la Figure 1.1 : la perception, la planification (ou la prise de décision) et le contrôle final à appliquer sur les actuateurs (les pédales d'accélération et de frein ainsi que l'angle au volant).

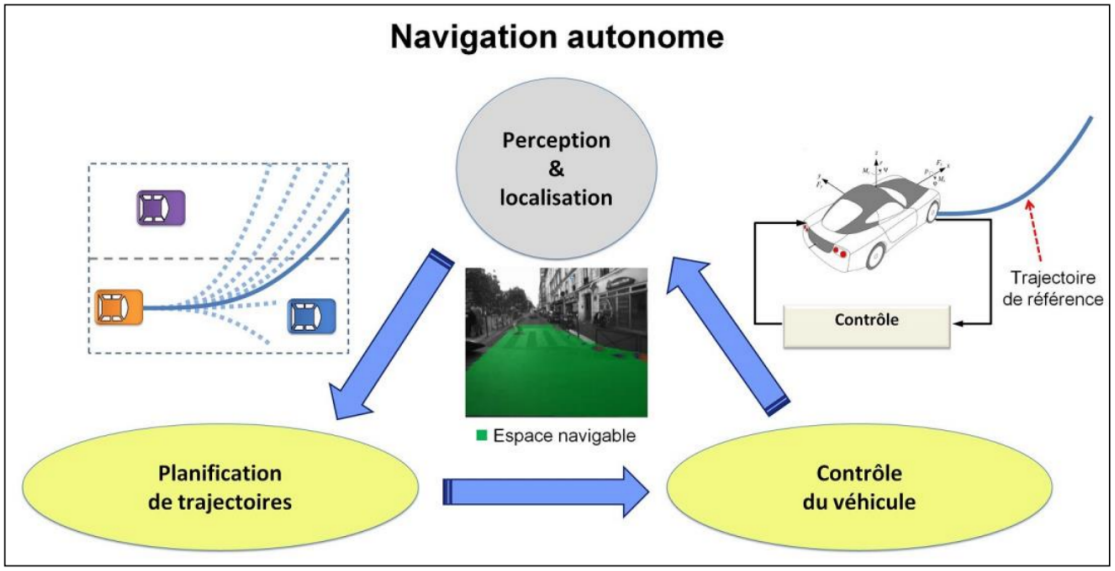


FIGURE 1.1 – Schéma illustrant l'architecture des systèmes de conduite autonome. Tout d'abord le module de perception rassemble les informations brutes provenant des différents capteurs placés sur le véhicule. Ensuite, des algorithmes de perception analysent ces données pour détecter les autres véhicules, les piétons, les lignes etc. Dans un deuxième temps, le bloc de planification prévoit les intentions des autres usagers de la route et, en fonction de celles-ci, calcule une trajectoire appropriée pour l'agent. Finalement, le bloc de contrôle prend en entrée cette trajectoire à suivre et calcule les commandes à appliquer au véhicule pour la suivre tout en respectant les contraintes physiques de la voiture ainsi que des contraintes de confort. Image provenant de Fokam [2014].

Dans le premier bloc de perception, les caméras, les LIDARs, les radars et les capteurs à ultrason intégrés balayent l'environnement autour du véhicule et des algorithmes de perception analysent les données brutes de ces capteurs pour détecter les autres véhicules, les piétons, les lignes etc. Ensuite, en utilisant les sorties du module de perception, le bloc de planification calcule une trajectoire appropriée pour l'agent. Finalement, le bloc de contrôle calcule les commandes à appliquer au véhicule pour suivre cette trajectoire le mieux possible.

Dans la majorité des approches actuelles, ces trois modules sont indépendants et seul le module de perception utilise des algorithmes d'apprentissage automatique.

Originellement, ces algorithmes de perception utilisaient des descripteurs fixes de l'image, comme les HOG (pour *Histogram of Oriented Gradients*) [Dalal and Triggs, 2005], permettant d'extraire des caractéristiques intéressantes des images d'entrée. Ces caractéristiques étaient ensuite données en entrée d'algorithmes d'apprentissage automatique pour faire de la classification ou de la détection d'objets par exemple.

Cependant, l'apprentissage profond a permis récemment d'améliorer considérablement les performances des algorithmes de perception. En effet, les réseaux de neurones permettent d'apprendre la tâche voulue directement à partir des données brutes (par exemple les images) et ne nécessitent plus d'utiliser des descripteurs fixes. Les réseaux de neurones vont en quelque sorte apprendre eux-mêmes quelles sont les caractéristiques importantes des données d'entrée. Ces avancées ont d'abord été appliqués à la classification d'image [Krizhevsky et al., 2012, Simonyan and Zisserman, 2014, He et al., 2016], à la segmentation d'image [Ronneberger et al., 2015] et à la détection d'objet [Girshick, 2015]. Ces méthodes ont ensuite été adaptés à d'autres représentations de données, en particulier la détection et la segmentation à partir de nuage de points 3D [Qi et al., 2017] provenant de capteurs LIDARs.

Les modules de planification et de contrôle quant à eux n'utilisent généralement pas d'apprentissage et reposent sur des modèles mathématiques explicites déterminés à priori : on dit que ces algorithmes sont fondés sur des règles. Cependant, ces règles sont nécessairement limitées et ne peuvent probablement pas s'adapter à la variabilité immense des situations possibles qui surviennent pour la voiture autonome, en particulier pour la conduite en environnement urbain. Utiliser des méthodes d'apprentissage semble donc nécessaire pour réussir à gérer toutes ces situations et cela nous amène à la question principale de cette thèse : comment apprendre les actions pour la voiture autonome ?

Il y a évidemment bien d'autres questions majeures qui restent à résoudre avant d'obtenir une véritable voiture autonome : la sécurité et l'explicabilité des algorithmes particulièrement ceux qui reposent sur de l'apprentissage automatique, la généralisation à des données en dehors de la distribution d'entraînement, la gestion des événements rares etc... mais ces questions ne seront pas traitées dans cette thèse.

1.2 Buts poursuivis par cette thèse

1.2.1 Objectif de la thèse

Comme mentionné précédemment, l'objectif principal de cette thèse est d'utiliser des méthodes d'apprentissage automatique pour apprendre les actions pour la voiture autonome.

Cette tâche est extrêmement complexe. En effet, prévoir et s'adapter aux comportements des autres usagers de la route peut parfois dépendre d'infimes informations comme le fait que le piéton semble pressé, occupé à quelque chose ou au contraire soit attentif à ce qui l'entoure. Les situations possibles d'interactions sont très variées, en particulier lorsque les règles du code de la route ne sont pas respectées. Les conducteurs humains sont capables de s'adapter incroyablement rapidement à des situations dangereuses alors

même qu'ils ne les ont jamais rencontrées mais cela reste pour le moment inaccessible aux algorithmes d'apprentissage. De plus, le fameux problème des *événements rares* est probablement encore plus présent pour le module de planification que pour le module de perception en particulier parce que l'aspect temporel est critique pour le premier alors qu'il est probablement moins important pour le deuxième. En effet, les actions à prendre peuvent grandement dépendre d'événements passés, par exemple si on voit le regard d'un cycliste, on sait qu'il nous a vu et cela va impacter nos décisions futures. La perception semble beaucoup plus orientée sur le moment présent même si, pour reprendre l'exemple précédent, détecter qu'une personne nous a vu semble aussi extrêmement difficile. De plus, comme mentionné précédemment, la perception a été considérablement améliorée par l'utilisation de méthodes d'apprentissage automatique qui utilisent de moins en moins de règles faites à la main : on laisse les algorithmes d'apprentissage apprendre eux-mêmes les caractéristiques importantes. Cette amélioration pourrait donc sûrement se produire aussi pour le choix des actions, i.e. ne plus utiliser des modèles fixés à priori mais laisser l'algorithme apprendre de lui-même le comportement à suivre. Nous verrons qu'il y a cependant de nombreux obstacles à cela et que ces obstacles sont la raison pour laquelle il n'y a actuellement quasiment jamais d'apprentissage dans les algorithmes de prise de décision pour la voiture autonome.

Toutes ces raisons plus le fait que l'immense majorité de la recherche en intelligence artificielle pour la voiture autonome se concentre sur le module de perception m'ont convaincu que le choix des actions serait probablement le dernier obstacle à résoudre avant d'atteindre un véhicule totalement autonome. Cette thèse va donc se concentrer essentiellement sur l'apprentissage des actions pour la voiture autonome.

1.2.2 Comment apprendre les actions ?

Un des axes de recherche le plus prometteur pour apprendre la décision est d'utiliser de l'apprentissage par renforcement profond (DRL pour *Deep Reinforcement Learning*).

L'apprentissage par renforcement (RL) est un des trois grands paradigmes de l'apprentissage automatique. Il se distingue de l'apprentissage supervisé par le fait que les agents apprennent par essai-erreur à partir d'un signal de récompense et non pas par simple supervision avec des paires entrée-label comme pour l'apprentissage supervisé, le type d'apprentissage le plus utilisé aujourd'hui dans les applications d'intelligence artificielle. Dans l'apprentissage par renforcement, on cherche explicitement à optimiser des séquences d'actions afin de maximiser le comportement à long terme. L'intérêt majeur du RL est que l'agent apprend de lui-même le comportement à suivre en explorant et en interagissant avec son environnement : on n'a donc pas besoin d'indiquer explicitement les actions à prendre.

L'apprentissage profond (DL), quant à lui, a permis récemment d'améliorer considérablement les performances dans de nombreuses applications d'intelligence artificielle. Que ce soit la perception comme mentionné précédemment, ou bien la gestion du langage (traduction, génération de texte) et même finalement l'apprentissage du comportement en étant combiné avec de l'apprentissage par renforcement : c'est cela qu'on nomme DRL.

Le DRL entraîne des agents directement à partir de données de grande dimension, en utilisant d'abord un réseau de neurones pour extraire des caractéristiques pertinentes des données d'entrées, et ensuite exploiter ces caractéristiques pour choisir les actions en suivant des algorithmes d'apprentissage par renforcement. L'idée du DRL est donc de combiner la puissance de l'apprentissage profond à extraire des informations pertinentes des données d'entrées, à la capacité de l'apprentissage par renforcement à optimiser le comportement d'un agent pour une certaine tâche. Cette thèse va donc s'intéresser particulièrement à l'apprentissage par renforcement et nous décrirons en détail ce type d'apprentissage dans le premier chapitre de cette thèse, Chapitre 2.

Le DRL est en fait un domaine extrêmement récent contrairement au RL qui lui est bien plus ancien. Les premiers véritables succès du domaine datent seulement de quelques années et ont largement profité des progrès de l'apprentissage profond. L'un des premiers algorithmes de DRL, Deep Q-Network (DQN) [Mnih et al., 2015] a ainsi réussi à apprendre à jouer à différents jeux Atari en utilisant seulement comme entrée les pixels de l'écran et comme signal de récompense le score global donné par le jeu. Il est important de noter que l'architecture du réseau et les hyperparamètres étaient exactement les mêmes d'un jeu Atari à un autre.

Le succès le plus connu et le plus médiatisé du DRL est sans aucun doute AlphaGo [Silver et al., 2016] qui a battu le champion du monde de Go à la surprise générale : les experts prédisaient encore plusieurs dizaines d'années avant d'arriver à ce niveau. L'algorithme a d'abord appris de manière supervisée à reproduire les coups des professionnels humains puis s'est amélioré par renforcement en jouant contre lui-même. Peu de temps après AlphaGo Zero [Silver et al., 2017] surpassait la version précédente en apprenant seulement par renforcement en jouant contre lui-même ! L'algorithme a donc dû apprendre à jouer à partir de rien, i.e. en jouant contre lui même de manière aléatoire. Suite à ces grands succès, de nombreux chercheurs ont essayé d'appliquer du DRL à différents domaines et en particulier à la conduite de voiture autonome. Mnih et al. [2016] ont ainsi démontré qu'un agent pouvait apprendre à conduire dans le jeu de course TORCS [Wymann et al., 2000]. L'avantage de l'apprentissage par renforcement en simulation est que l'agent peut réellement rencontrer des situations d'échecs comme heurter des piétons ou rentrer dans les murs et apprendre ensuite comment éviter ou récupérer d'une mauvaise situation. Cette thèse va principalement s'intéresser à l'application de l'apprentissage par renforcement pour l'apprentissage des actions pour la voiture autonome.

Un autre axe de recherche possible pour l'apprentissage des actions est l'apprentissage par imitation, c'est à dire essayer d'apprendre à conduire en observant comment un expert, i.e. un conducteur humain, agit. Cet axe sera moins étudié lors de cette thèse qui se concentre principalement sur l'apprentissage par renforcement. Cependant, le Chapitre 6 y sera entièrement consacré. Dans ce chapitre, nous détaillerons plus en détail ce qu'est l'apprentissage par imitation et nous présenterons notre premier travail intégré sur un véhicule réel [Toromanoff et al., 2018] fondé sur de l'apprentissage par imitation.

Finalement, nous avons fait le choix dans cette thèse de ne pas utiliser une approche

modulaire classique (c.f. Figure 1.1) consistant à développer trois modules indépendants pour la perception, la planification et le contrôle. Nous avons opté pour une approche de bout-en-bout (en anglais *end-to-end*), c'est à dire prendre les données brutes des capteurs en entrée d'un bloc unique qui calcule directement les contrôles à appliquer sur le véhicule. L'intuition est toujours de limiter au maximum les à priori et de laisser l'algorithme apprendre de lui-même quels sont les éléments pertinents des données d'entrée plutôt que de passer par une représentation haut niveau de la scène. Un exemple d'une telle représentation haut niveau pourrait être la position et la vitesse de chacun des objets de la scène, mais cela utilise l'à priori que seul les objets sont pertinents. On peut en fait imaginer des cas où d'autres éléments sont pertinents, par exemple si il pleut ou qu'il neige, cela peut totalement changer le comportement à suivre car la physique de la route sera différente. Une explication détaillée des différentes représentations haut niveau possibles de la scène ainsi que des avantages et inconvénients d'utiliser une approche de bout-en-bout peut se trouver dans le Chapitre 2, Section 2.5.2. En pratique, dans l'ensemble de ce manuscrit, nous avons toujours utilisé les images d'une unique caméra frontale comme seule entrée de notre algorithme de bout-en-bout.

1.3 Cadre de la thèse

Cette thèse de trois ans a été effectuée avec le centre de Robotique (CAOR) de Mines ParisTech ainsi qu'avec l'entreprise Valeo. Ces deux entités ont une longue histoire de collaborations via différents projets collaboratifs ou bien le financement d'étudiants en doctorat.

Le laboratoire CAOR est spécialisé en robotique et s'intéresse principalement aux questions de perception, de localisation, de planification et de contrôle de trajectoire. Ce laboratoire s'intéresse ainsi depuis longtemps à l'application de la voiture autonome, par exemple du traitement d'image pour la détection de panneaux de signalisation [Moutarde et al., 2007] ou bien la gestion optimisée du traffic sur les intersections [Qian et al., 2014, 2015].

Valeo quant à lui est un équipementier automobile qui, entre autres, produit des capteurs comme des caméras, des LIDARs, des radars et qui développe aussi des algorithmes et des logiciels pour faire de la perception à partir des données issues de ces capteurs. Cette thèse a été initiée par l'équipe du *Driving Assistance Research* (DAR) qui se concentre essentiellement sur la recherche et le développement pour la voiture autonome et l'aide à la conduite. Le DAR comporte environ 100 ingénieurs en recherche et développement et possède plusieurs véhicules d'essais équipés de différents capteurs pour la collecte de données et les tests réels. J'ai aussi rejoint Valeo.AI après sa création en 2018. Ce laboratoire industriel de recherche travaille et publie principalement dans les domaines de la perception : de la fusion de données provenant de différents capteurs à l'adaptation de domaine et l'estimation d'incertitude.

1.4 Structure du manuscrit

Le reste de ce manuscrit est organisé comme suit.

Dans le Chapitre 2, nous allons d'abord introduire les différents algorithmes et les bases de l'apprentissage par renforcement. Nous y détaillerons ensuite comment ces algorithmes ont été associés avec de l'apprentissage profond pour donner les premiers algorithmes d'apprentissage par renforcement profond. Finalement, nous exposerons notre approche pour appliquer de l'apprentissage par renforcement à la voiture autonome, en particulier quelle représentation d'entrée nous allons utiliser ainsi que la famille d'algorithme de renforcement qui semble la plus adaptée pour notre cas d'application.

Le Chapitre 3 présente un travail préliminaire à cette thèse dont l'objectif était d'apprendre à conduire dans un jeu de course avec des graphismes et une physique réaliste. Ce travail a été effectué en amont de cette thèse et a permis de soulever de nombreuses problématiques qui seront abordées dans la suite de ce manuscrit.

Dans le Chapitre 4, nous présenterons le premier travail véritablement effectué durant cette thèse, de l'apprentissage par renforcement appliqué au benchmark Atari. Dans ce chapitre, notre but est de développer le meilleur algorithme général, i.e. réussissant à obtenir les meilleures performances possibles sur la grande diversité des jeux Atari en n'utilisant aucune information spécifique. L'idée est que si un algorithme est capable d'apprendre des tâches variées avec les mêmes hyperparamètres et architecture, cet algorithme devrait pouvoir être transférable facilement et être performant sur n'importe quelle autre tâche, comme la conduite autonome. Nous avons aussi dû faire face à des problèmes pour faire des comparaisons équitables ou reproduire les travaux précédents et nous avons donc introduit un nouveau standard d'entraînement et d'évaluation sur le benchmark Atari permettant une meilleure reproductibilité des résultats. Finalement, nous avons implémenté une architecture distribuée de notre algorithme permettant d'utiliser bien plus de ressources de calcul en parallèle et qui permet ainsi de faire des entraînements en un temps raisonnable même si la génération des expériences est lente.

Le Chapitre 5 s'intéresse au problème du contrôle d'un véhicule dans un environnement urbain ce qui inclut le maintien sur sa voie, la détection des feux tricolores ainsi que la gestion de trafic aux intersections. Cette tâche est particulièrement difficile et nous avons montré pour la première fois un algorithme d'apprentissage par renforcement réussissant à conduire correctement dans un environnement aussi complexe. Pour cela nous avons séparé l'entraînement de nos agents en deux phases. Tout d'abord, une phase d'entraînement supervisé de différents indices utiles à la conduite comme l'estimation d'une carte de segmentation ou de l'état du feu tricolore. Dans une deuxième phase, on utilise les features provenant du réseau pré-entraîné, qui est fixé durant l'entraînement par renforcement, comme entrée pour notre algorithme de renforcement. Nous avons nommé cette technique *indices implicites* car l'agent n'utilise pas les indices explicites mais seulement les features qui permettent à notre premier réseau d'estimer les différents indices. Pour l'algorithme de renforcement lui même, nous avons utilisé la même architecture distribuée que celle développée auparavant pour Atari.

Pour introduire les applications au véhicule réel, nous détaillerons dans le Chapitre 6

ce qu'est l'apprentissage par imitation ainsi qu'un état de l'art de l'apprentissage par imitation appliqué à la voiture autonome. Nous exposerons aussi notre propre travail d'apprentissage par imitation intégré sur un véhicule réel pour le contrôle du volant. L'idée fondamentale de ce travail est de réussir à générer automatiquement des situations d'échec et leurs corrections associées pour que l'agent appris par imitation sache réagir à ses propres erreurs.

Le dernier chapitre de cette thèse, Chapitre 7, expose des travaux encore en cours consistant à appliquer de l'apprentissage par renforcement sur un véhicule réel pour le contrôle dans un environnement urbain. Pour cela nous nous sommes inspirés d'un travail très récent, l'article VISTA [Amini et al., 2020] qui est le premier article (et le seul à ce jour) à avoir transféré sur un véhicule réel un modèle appris par renforcement en simulation sans ré-apprentissage dans le monde réel. Nous avons aussi grandement utilisé nos propres travaux précédents, en particulier notre technique d'*indices implicites* qui permet d'entraîner des réseaux de neurones avec plus de paramètres et avec des entrées de bien plus grande dimension que ceux utilisés par les travaux précédents en DRL. Il n'y a encore aucun travail publié qui applique du renforcement à la conduite en environnement urbain et nos résultats pour le contrôle du volant sont extrêmement prometteurs. Cependant, nos résultats pour le contrôle longitudinal (accélération et frein) ne sont pas encore assez matures et il reste de nombreux points à résoudre avant de pouvoir réellement intégrer ce contrôle longitudinal dans une vrai voiture.

Finalement, nous faisons un résumé de nos principales contributions ainsi qu'un aperçu des travaux futurs possibles dans la conclusion de cette thèse, Chapitre 8.

Chapitre 2

Apprentissage par renforcement et application au véhicule autonome

Dans ce chapitre, nous introduirons d'abord la théorie de l'apprentissage par renforcement. Puis, nous détaillerons comment associer des réseaux de neurones à de l'apprentissage par renforcement. Finalement, nous présenterons plus spécifiquement l'application du renforcement au contrôle d'un véhicule autonome.

Dans un premier temps, nous décrirons ce qu'est l'apprentissage par renforcement en introduisant les bases et différentes notations et fonctions qui seront utiles tout au long de ce manuscrit. Ensuite, nous présenterons les 3 grandes familles d'algorithmes de renforcement, tout d'abord en explicitant la théorie sur laquelle repose chacune de ces familles puis en introduisant certains algorithmes représentatifs. Nous parlerons ensuite plus spécifiquement de l'état de l'art du renforcement appliqué à la voiture autonome pour finalement conclure sur les choix techniques et d'application de cette thèse.

Toute cette partie a été grandement inspirée par 2 sources très intéressantes et complètes sur le RL, tout d'abord le livre de R. Sutton, *Reinforcement Learning : an introduction* [Sutton and Barto, 2018] (deuxième version mise à jour en 2017) et aussi le cours CS294 [Levine, 2017] de Sergey Levine à Berkeley intitulé *Deep Reinforcement Learning*.

2.1 Bases de l'apprentissage par renforcement

Quand on pense apprentissage, on a souvent en tête l'apprentissage d'un bébé qui découvre le monde. En effet, un bébé à la naissance ne possède quasiment aucune connaissance du monde qui l'entoure ni même de son propre corps ou de ses propres mouvements. Il agit initialement de manière quasi aléatoire puis à force d'expériences s'améliore pour finalement réussir à atteindre certains buts qui lui étaient complètement étrangers à sa naissance comme marcher, parler etc... En suivant une stratégie d'*essai-erreur* et en interagissant avec son environnement, le bébé peut développer de nouvelles compétences (ce mécanisme est identique pour l'apprentissage des animaux). L'apprentissage par renforcement est le domaine de l'apprentissage automatique qui s'intéresse spécialement à ce paradigme et qui formalise mathématiquement cette situation.

Dans le renforcement, on part de l'hypothèse que l'agent peut agir dans son environnement via ce que l'on appellera *actions* tout le long de cet exposé. Pour choisir ces actions, l'agent peut observer l'état du monde qui l'entoure soit de manière totale si il a

accès à toutes les informations, soit de manière partielle (on parle alors d'environnement partiellement observable). On appellera *état* les informations disponibles à l'agent et on traitera indifféremment les cas totalement et partiellement observables même si dans certaines situations cela peut avoir une importance. De plus, à chaque action effectuée, l'agent reçoit un scalaire indiquant si l'action a été bénéfique ou non, on appellera *récompense* ce signal d'apprentissage dans la suite de ce manuscrit. Le but final de l'agent est de maximiser la somme des récompenses cumulées de l'instant initial à l'instant final.

Cette approche se distingue à la fois de l'apprentissage supervisé où l'on indique explicitement ce que devrait faire l'agent à chaque instant, et à la fois de l'apprentissage non supervisé où il n'y a aucun signal d'apprentissage à proprement parler et où le but est d'essayer de trouver des structures et des relations dans les données d'entrée. Ainsi, le renforcement est un troisième paradigme de l'apprentissage automatique. Une autre hypothèse très importante dans le renforcement est que chaque action influe non seulement sur la récompense instantanée mais aussi sur l'état suivant et à travers cela sur toutes les récompenses subséquentes. C'est pour cela qu'intrinsèquement, le renforcement s'intéresse à optimiser des *séquences d'actions* d'un agent dans l'environnement.

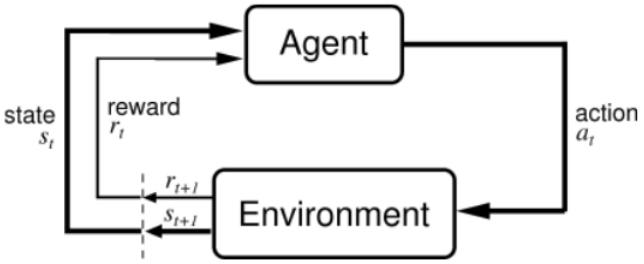


FIGURE 2.1 – Schéma représentant l'interaction entre l'agent et l'environnement. En fonction de l'état s_t (*state* en anglais), l'agent choisit une action a_t effectuée dans l'environnement, ce qui amène dans un nouvel état s_{t+1}, et apporte une récompense r_{t+1} (*reward* en anglais). Image provenant de Sutton and Barto [2018].

Plus spécifiquement, l'agent et l'environnement interagissent à chaque instant d'une séquence de temps discret $t = 0, 1, 2....$ A chaque pas de temps t, l'agent reçoit une représentation de l'état de l'environnement $s_t \in S$ et il doit choisir une action $a_t \in A$ en prenant en compte cet état. Au pas de temps $t + 1$, l'agent reçoit une récompense $r_{t+1} \in \mathbb{R}$ ainsi que le nouvel état de l'environnement s_{t+1} comme illustré Figure 2.1.

Le problème de l'apprentissage par renforcement peut être formalisé mathématiquement par un processus de décision markovien (MDP pour *Markovian Decision Process*). Un processus de décision markovien est un tuple de 5 termes (S, A, P, r, γ) où :

— S est l'espace des états de l'environnement (généralement fini)

— A est l'espace des actions possibles (discret ou continu)

— P est la probabilité de transition d'un état à un autre. $p(s'|s, a)$ est la probabilité d'arriver dans l'état s' en effectuant l'action a dans l'état s

— r est la fonction de récompense : $r(s, a) \in \mathbb{R}$ est la récompense obtenue en effectuant l'action a dans l'état s

— $\gamma \in [0, 1]$ est le facteur de dévaluation qui représente la différence d'importance entre les récompenses à plus ou moins long terme : si γ égal 0, on ne cherche qu'à optimiser la récompense instantanée tandis que si il vaut 1, une récompense future est aussi importante qu'une récompense instantanée

Une trajectoire τ est une séquence $s_0, a_0, s_1, a_1, ...s_T, a_T$. Le but de l'agent est de maximiser la somme des récompenses cumulées $R_t = \sum_{k=0}^{T} \gamma^k r_{t+k}$ obtenues lors de la trajectoire associée. On note que lorsque T est fini (i.e. $T < +\infty$) on parle d'épisode fini tandis que si T est infini, on parle d'épisode continu (dans ce cas on prend $\gamma < 1$).

2.2 Politique et fonction de valeur

Une *politique* est une fonction $\pi : S \to A$ qui associe à chaque état une densité de probabilité sur l'espace des actions. S'il n'y a qu'une action associée à chaque état on parle de politique déterministe. Dans le cas général, on note $\pi(a_t|s_t)$ la probabilité de prendre l'action a_t dans l'état s_t.

La grande majorité des algorithmes de RL estime aussi ce que l'on dénomme une *fonction de valeur* qui est une fonction estimant à quel point un état est intéressant, i.e. quelle est la somme des récompenses cumulées que l'on peut espérer à partir de cet état. Cela dépend bien évidemment des actions choisies, c'est pour cela que par définition une fonction de valeur V_π est toujours associée à une politique π.

Pour le formuler mathématiquement, la valeur $V_\pi(s)$ d'un état s est l'espérance de la somme des récompenses cumulées en partant de l'état s et en suivant la politique π jusqu'à la fin de l'épisode.

$$V_\pi(s) = \mathbb{E}_\pi[R_t|s_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{T} \gamma^k r_{t+k}|s_t = s] \tag{2.1}$$

où $\mathbb{E}_\pi$ dénote l'espérance sur les trajectoires possibles en suivant la politique π pour le choix des actions.

On définit aussi la *Q-fonction* qui correspond à la valeur en partant d'un état s et en choisissant l'action a (qui peut être différente de l'action choisie par la politique π) puis en suivant la politique π jusqu'à atteindre un état terminal.

$$Q_\pi(s, a) = \mathbb{E}_\pi[R_t|s_t = s, a_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{T} \gamma^k r_{t+k}|s_t = s, a_t = a] \tag{2.2}$$

On appelle V_π la fonction état-valeur et Q_π la fonction action-valeur.

Il est important de noter que les fonctions de valeurs peuvent s'exprimer itérativement en fonction des valeurs aux états suivants. En effet :

$$V_\pi(s) = \mathbb{E}_\pi[r_t + \gamma R_{t+1}] = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)[r(s,a) + \gamma V_\pi(s')] \tag{2.3}$$

L'expression 2.3 est l'équation de Bellman pour V_π (on peut obtenir une forme similaire pour la Q-fonction). L'équation de Bellman est à la base de nombreuses méthodes pour apprendre à estimer V_π comme nous le verrons dans la suite de ce manuscrit.

Ces trois notions, politique, fonction état-valeur et fonction action-valeur, seront à la base de tous les algorithmes de RL que l'on présentera par la suite. En effet, résoudre un problème de RL consiste à trouver une politique qui permet d'obtenir de grandes récompenses sur le long terme. Pour les processus de décision markovien finis (on sera toujours dans ce cas dans ce manuscrit), on peut précisément définir une politique optimale grâce aux fonctions de valeurs qui définissent un ordre partiel sur l'ensemble des politiques. Une politique π est définie comme étant meilleure ou égale à une politique π' si le retour espéré est plus grand pour chaque état s en suivant π plutôt que π', i.e $\pi \geqslant \pi' si\ \forall s \in S, V_\pi(s) \geqslant V_{\pi'}(s)$.

Il existe toujours au moins une politique déterministe qui est meilleure ou égale à toute autre politique. On appelle cela une politique optimale π^*, à noter qu'il peut y avoir plusieurs politiques optimales mais elles ont toutes la même fonction de valeur V^* et on les dénote toutes par π^*. Tous les algorithmes qui suivront cherchent à estimer cette politique optimale d'une manière ou d'une autre et partagent tous la même architecture globale présentée sur le schéma Figure 2.2.

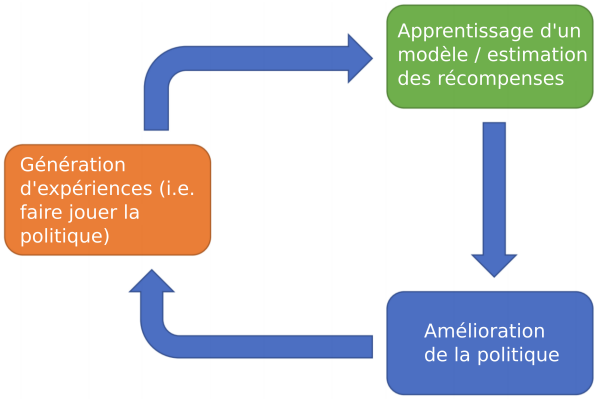


FIGURE 2.2 – Architecture générale d'un algorithme d'apprentissage par renforcement. Image adaptée de Levine [2017].

2.3 Les familles d'algorithmes d'apprentissage par renforcement

Comme dit précédemment, le but de tout algorithme de RL est d'estimer la politique optimale π^*, mais ces algorithmes peuvent se catégoriser en trois grandes familles.

Dans cette sous-partie nous allons présenter séparément ces trois familles, d'abord en expliquant la théorie sur laquelle elles reposent, puis en détaillant quelques algorithmes principaux de chaque famille. Il est important de noter que nous ne nous intéressons pas encore à l'association du renforcement avec des réseaux de neurones, ce sujet sera traité dans la prochaine section 2.4.

2.3.1 Fondés sur la fonction de valeur

La première famille d'algorithmes que nous allons présenter cherche à estimer la fonction de valeur de la politique optimale. Nous allons d'abord introduire la notion de Programmation Dynamique qui sera à la base de l'ensemble des algorithmes que nous décrirons dans cette partie.

2.3.1.1 Programmation dynamique

La programmation dynamique fait référence à des algorithmes qui peuvent être utilisés pour calculer la politique optimale en admettant que l'on connaît parfaitement le modèle de l'environnement. On dit que le modèle est connu lorsque la fonction de transition ainsi que la fonction de récompense sont connues. Ces algorithmes sont généralement inutilisables en pratique car ils partent de l'hypothèse d'une représentation parfaite du modèle et sont très peu efficaces. Cependant, ils permettent d'avoir une bonne intuition des algorithmes qui suivront.

Nous introduirons d'abord comment évaluer la fonction de valeur d'une politique arbitraire π. On appelle cela le problème de *prédiction*, à différencier du problème du *contrôle* où l'on cherche à trouver la politique optimale π^*. La programmation dynamique permet de trouver V_π itérativement en partant d'une fonction V_0 arbitraire.

$$\forall s \in S,\ V_{k+1}(s) = r(s, \pi(s)) + \gamma \sum_{s'} p(s'|s, a) V_k(s')\ (\textit{prédiction}) \tag{2.4}$$

On sait que $V_k = V_\pi$ est un point fixe de cette règle de mise à jour (voir l'équation de Bellman 2.3 pour V_π). Et on peut démontrer que V_k tend en fait vers V_π quand k tend vers l'infini. On constate qu'on estime la valeur d'un état en utilisant l'équation de Bellman comme une équation de mise à jour. Le fait d'utiliser l'estimation de la valeur des états suivants pour estimer la valeur de l'état actuel se nomme *boostrapper*.

La deuxième idée est maintenant d'améliorer la politique sachant que l'on a calculé sa fonction de valeur, i.e. le problème du *contrôle*. Pour cela on va définir une politique π' telle que :

$$\forall s \in S,\ \pi'(s) = \arg\max_{a \in A} (r(s, a) + \gamma \sum_{s'} p(s'|s, a) V_\pi(s'))\ (\textit{contrôle}) \tag{2.5}$$

Cette politique π' est égale ou meilleure que la politique π initiale et on peut même démontrer que si elles sont égales ($\pi' = \pi$) alors c'est que l'on a déjà atteint la politique

optimale ($\pi = \pi^*$). C'est pour cela que ce processus converge vers la politique optimale. Par contre ce processus est extrêmement inefficace car il faut alterner un pas d'évaluation de politique jusqu'à convergence (à un epsilon fixé près) avec un pas d'amélioration de politique et cela jusqu'à convergence finale vers la politique optimale.

L'algorithme *itération de valeur* permet de combiner ces deux pas en un seul pour être beaucoup plus efficace.

$$\forall\ s \in S, V_{k+1}(s) = \max_a(r(s,a) + \gamma \sum_{s'} p(s'|s,a)V_k(s'))\ (\textit{itération de valeur}) \tag{2.6}$$

Cet algorithme converge vers la fonction de valeur optimale V^* d'où l'on peut déduire la politique optimale en prenant l'argument maximum sur les actions des valeurs aux états suivants.

$$\forall\ s \in S, \pi^*(s) = \arg\max_{a \in A}(r(s,a) + \gamma \sum_{s'} p(s'|s,a)V^*(s')) \tag{2.7}$$

Le problème de tous ces algorithmes est qu'ils sont totalement insolubles si le modèle n'est pas connu ou même juste si l'espace d'états est trop grand car il faut parcourir tous les états un par un. Pour pallier ce défaut, des méthodes utilisant des échantillons ont été développées, mais qui sur le principe essayent de réappliquer les idées introduites par la programmation dynamique.

2.3.1.2 Méthode de Monte-Carlo

Si le modèle est inconnu, l'idée la plus simple pour estimer la fonction de valeur d'une politique arbitraire π (*prédiction*) est de jouer cette politique de nombreuses fois en gardant en mémoire les états rencontrés et la somme des récompenses obtenues à partir de ces états. D'après la loi des grands nombres, la moyenne des récompenses obtenues à partir d'un même état s convergera vers $V_\pi(s)$. Cette idée est la base de toutes les méthodes de Monte-Carlo.

$$\hat{V}_\pi(s_t) \leftarrow \hat{V}_\pi(s_t) + (1/n)(R_t^n - \hat{V}_\pi(s_t)) \tag{2.8}$$

A noter que dans l'équation 2.8, R_t^n est la somme des récompenses empiriques (et non plus l'espérance) de la n-ème trajectoire et n est l'index de la trajectoire en cours.

On constate que les estimateurs pour chaque état sont indépendants, on n'utilise pas l'estimation de la valeur des états suivants pour mettre à jour la valeur de l'état actuel. Autrement dit, Monte Carlo ne fait pas de *boostrapping*.

Lorsqu'un modèle de l'environnement n'est pas disponible, il est particulièrement utile d'estimer la fonction action-valeur plutôt que la fonction état-valeur. En effet, sans connaître le modèle, la fonction état-valeur n'est pas suffisante pour déterminer une politique car on ne sait pas dans quel état on sera après chaque action. Par contre,

connaître directement la Q-fonction permet de déterminer une meilleure politique en prenant l'argument maximal sur les actions, on parle de politique *gloutonne* (en anglais *greedy*) par rapport à cette Q-fonction (*contrôle*). L'idée est donc d'estimer la Q-fonction par Monte-Carlo : au lieu de juste garder en mémoire les états traversés, on garde en mémoire les paires état-action. Un point très important, et qui sera développé beaucoup plus dans le reste de ce manuscrit, est qu'il faut que chaque paire état-action soit visitée une infinité de fois. Si la politique est déterministe par exemple et que l'on prend toujours la même action dans un même état, on ne peut pas évaluer la Q-fonction pour toutes les autres actions et donc on ne pourra jamais améliorer la politique : les méthodes de Monte Carlo n'auront aucun échantillon à moyenner. Cela introduit l'idée d'exploration, i.e. il faut que toutes les actions soient possibles avec une probabilité non nulle. L'idée la plus simple pour effectuer cela est d'utiliser une politique $\epsilon - greedy$, i.e. une probabilité ϵ de prendre une action aléatoire et le reste du temps juste prendre l'action donnée par la politique gloutonne dérivée de l'estimation actuelle de la Q-fonction.

2.3.1.3 Méthode de différence temporelle

L'apprentissage par différence temporelle est une combinaison des idées des méthodes de Monte Carlo avec celles de la Programmation Dynamique. Ces méthodes utilisent des échantillons (comme les méthodes de Monte Carlo) et ne nécessitent pas de connaître le modèle de l'environnement. Comme pour la programmation dynamique, elles mettent à jour les fonctions de valeur en utilisant des estimations de la valeur des états suivants. Cela diffère des méthodes de Monte Carlo qui nécessitent d'attendre la fin de l'épisode pour mettre à jour les paramètres.

$$\underbrace{V_\pi(s_t) = \mathbb{E}_\pi[R_t]}_{\text{1 : Monte Carlo}} \underbrace{= \mathbb{E}_\pi[r_{t+1} + \gamma V_\pi(s_{t+1})]}_{\text{2 : Programmation Dynamique}} \tag{2.9}$$

Les méthodes de Monte Carlo estime la fonction de valeur en utilisant la première égalité et en approximant R_t avec des échantillons. La programmation dynamique estime la fonction de valeur en utilisant la deuxième égalité : elle a accès à la vraie récompense r_{t+1} car le modèle est connu mais elle fait du *boostrapping*, i.e. la fonction de valeur aux états suivants n'est qu'une estimation. L'apprentissage par différence temporelle utilise ces deux idées ensemble, i.e. utiliser la récompense échantillonnée en plus d'utiliser l'estimation actuelle de la valeur de l'état suivant.

Comme précédemment nous allons d'abord présenter une méthode pour le problème de la *prédiction*, i.e. pour évaluer la fonction de valeur d'une politique arbitraire π. L'algorithme le plus simple d'apprentissage par différence temporelle est le TD(0) ou *one-step TD* (TD pour Temporal Difference) dont la règle de mise à jour est la suivante :

$$\hat{V}_\pi(s_t) \leftarrow \hat{V}_\pi(s_t) + \alpha(r_{t+1} + \gamma \hat{V}_\pi(s_{t+1}) - \hat{V}_\pi(s_t)) \tag{2.10}$$

Avec α le pas d'apprentissage : $\alpha = 1/n$ a des propriétés de convergence théoriques mais en pratique on prend souvent $\alpha =$ constante qui ne converge pas théoriquement.

Cette règle de mise à jour possède l'avantage de ne pas dépendre d'un modèle de l'environnement (contrairement à la Programmation Dynamique) et de ne pas attendre la fin d'un épisode pour faire des mises à jour (contrairement aux méthodes de Monte-Carlo). Nous allons finalement appliquer ces idées au problème du *contrôle* et ces algorithmes seront à la base de la grande majorité des algorithmes de l'état de l'art.

Comme évoqué brièvement pour les méthodes de Monte Carlo, la première étape est d'apprendre une fonction action-valeur plutôt qu'une fonction état-valeur (car on ne connaît pas le modèle) et pour cela il y a un compromis à faire entre exploration et exploitation pour pouvoir assurer que chaque paire état-action soit visitée une infinité de fois. Pour cela, on maintient une estimation $\hat{Q}^*$ de la Q-fonction optimale et l'on choisit les actions avec une politique $\epsilon - greedy$ dérivée de l'estimation actuelle de la Q-fonction optimale pour assurer un minimum d'exploration. Mettre à jour la Q-fonction peut se faire de manière similaire qu'apprendre la fonction de valeur V_π, sauf qu'on prend en compte la paire état-action et non plus seulement l'état. La règle de mise à jour devient donc :

$$\hat{Q}^*(s_t, a_t) \leftarrow \hat{Q}^*(s_t, a_t) + \alpha(r_{t+1} + \gamma \hat{Q}^*(s_{t+1}, a_{t+1}) - \hat{Q}^*(s_t, a_t)) \tag{2.11}$$

Cet algorithme (voir Figure 2.3) utilise le quintuplet $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, c'est pour cela qu'il a le nom de SARSA [Rummery and Niranjan, 1994].

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

FIGURE 2.3 – Algorithme *on-policy* SARSA . Image provenant de Sutton and Barto [2018].

Il est important de noter que cet algorithme est *on-policy*, c'est à dire que la politique utilisée pour effectuer les actions (la politique *comportementale*) est la même que la politique que l'on cherche à estimer. L'algorithme SARSA met à jour la fonction de valeur en utilisant l'état suivant ainsi que l'action qu'aurait prise la politique actuelle à cet état suivant. Autrement dit, il estime les récompenses attendues en supposant que la politique actuelle continue à être effectuée. On peut démontrer que SARSA converge vers la Q-fonction optimale. Si l'on veut être exact, les actions étant toujours prises de manière $\epsilon - greedy$, SARSA converge vers la politique optimale prenant en compte le facteur aléatoire de l'exploration.

Il existe un autre algorithme, l'algorithme du Q-Learning [Watkins and Dayan, 1992], très similaire mais qui lui estime réellement la politique optimale qui est donc une politique déterministe. La différence fondamentale est que c'est un algorithme *off-policy* : la politique *comportementale* peut être différente de la politique qu'on cherche à estimer. Ainsi en théorie on peut même déterminer la politique optimale en exécutant seulement une politique aléatoire dans l'environnement. L'idée est que la politique comportementale doit continuellement explorer (l'exemple le plus commun étant $\epsilon - greedy$ par rapport à l'estimation de la Q-fonction actuelle) mais la politique estimée est elle déterministe car c'est la politique optimale. La règle de mise à jour est la suivante :

$$\hat{Q}^*(s_t, a_t) \leftarrow \hat{Q}^*(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a \in A} \hat{Q}^*(s_{t+1}, a) - \hat{Q}^*(s_t, a_t)) \tag{2.12}$$

Il est important de noter que l'algorithme du Q-Learning ne s'applique qu'à un espace d'actions discret, à cause du *max* sur l'ensemble des actions dans la mise à jour.

La Figure 2.4 illustre la différence entre les politiques optimales trouvées par un algorithme on-policy et un algorithme off-policy si la politique comportementale est $\epsilon - greedy$ avec $\epsilon > 0$.

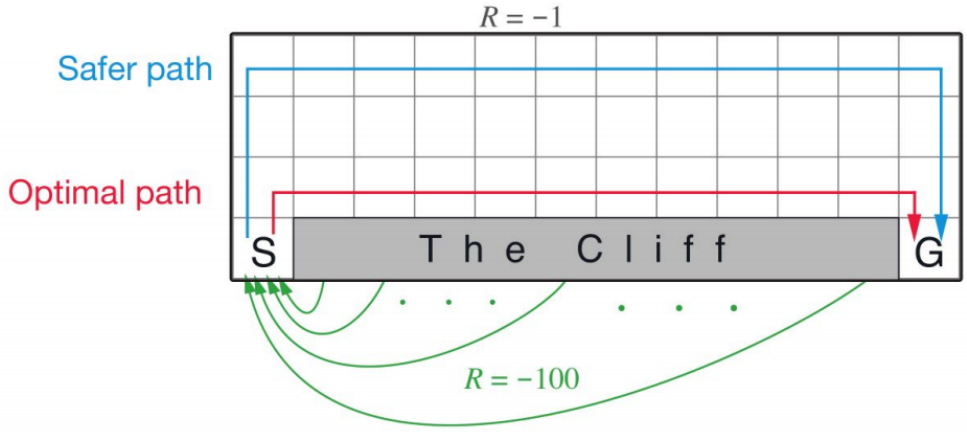


FIGURE 2.4 – Politiques optimales trouvées par Q-learning *off-policy* (rouge) et SARSA *on-policy* (bleu) pour le problème du ravin. Le but étant d'atteindre l'objectif G le plus rapidement possible (récompense de -1 à chaque pas de temps) tout en évitant de tomber dans le ravin (récompense de -100 et retour à l'état initial S). Comme SARSA est un algorithme on-policy, il prend en compte le caractère aléatoire de l'exploration et s'éloigne donc le plus possible du ravin alors que Q-Learning quant à lui converge réellement vers la politique optimale et le chemin le plus court.

En fait on peut relier les méthodes de Monte-Carlo avec les méthodes de différence temporelle en introduisant le principe de *multi-step TD*. L'idée est d'utiliser plusieurs pas de récompenses échantillonnées avant de bootstrapper. La règle de mise à jour devient donc :

$$\hat{V}_\pi(s_t) \leftarrow \hat{V}_\pi(s_t) + \alpha(\sum_{t'=t+1}^{t'=t+n} \gamma^{t'-t-1} r_{t'} + \gamma^n \hat{V}_\pi(s_{t+n}) - \hat{V}_\pi(s_t)) \tag{2.13}$$

Si n = 1, on retrouve le *one-step TD* (équation 2.10) et si on prend n = ∞ on obtient les méthodes de Monte-Carlo (équation 2.8). De manière générale, ni le one step, ni Monte Carlo ne sont optimaux et il vaut mieux en pratique utiliser un nombre intermédiaire de pas pour atteindre les meilleurs performances.

2.3.2 Fondés sur la politique

2.3.2.1 Algorithme du gradient de la politique

La deuxième famille d'algorithmes que nous allons présenter sont des algorithmes qui cherchent directement à optimiser et estimer la politique optimale, d'où le nom de *fondés sur la politique*. Pour cela, ces algorithmes utilisent une politique π_θ avec des paramètres θ et cherchent à optimiser ces paramètres. On peut formaliser mathématiquement cela comme chercher les paramètres optimaux θ^* tel que :

$$\theta^* = \arg\max_\theta \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[\sum_t \gamma^t r(s_t, a_t)] = \arg\max_\theta J(\theta) \tag{2.14}$$

L'idée est donc d'optimiser cet objectif par remontée de gradient.

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[\underbrace{r(\tau)}_{\sum_t \gamma^t r(s_t,a_t)}] = \int \pi_\theta(\tau) r(\tau)\, \mathrm{d}\tau \tag{2.15}$$

$$\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau) r(\tau)\, \mathrm{d}\tau = \int \pi_\theta(\tau) \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} r(\tau)\, \mathrm{d}\tau \tag{2.16}$$

$$= \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)] \tag{2.17}$$

Or $\pi_\theta(s_0, a_0, ..., s_T, a_T) = p(s_0) \prod_{t=0}^T \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$

Donc $\nabla_\theta \log \pi_\theta(\tau) = \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t|s_t)$

On obtient donc $\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[(\sum_{t=0}^T \frac{\nabla_\theta \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)})(\sum_{t=0}^T \gamma^t r(s_t, a_t))]$

Ceci est le théorème du gradient de la politique. Cette expression étant une espérance sur les trajectoires obtenues en effectuant la politique π_θ, on peut en obtenir une approximation via Monte-Carlo avec des échantillons de trajectoires. Cela est à la base de l'algorithme REINFORCE [Williams, 1992] (voir Figure 2.5), qui est l'un des premiers algorithmes *fondés sur la politique*.

On peut voir dans cette expression une mise à jour intuitive des paramètres de la politique. En effet, on met à jour les paramètres proportionnellement à la somme des récompenses multipliée par le gradient de la probabilité de prendre les actions et inversement proportionnellement à la probabilité de ces actions. L'idée est de rendre plus fréquentes les actions ayant contribué à de plus grandes récompenses tout en contrebalançant l'avantage des actions les plus probables. En effet, si une action a très peu de chance d'être choisie par la politique actuelle mais qu'elle a abouti à une très grande récompense, on veut tendre à la rendre plus probable qu'une action commune aboutissant à une récompense moindre.

Une autre information importante est d'utiliser le principe de causalité, i.e. que les actions prises à un instant t ne peuvent pas modifier les récompenses aux instants t' avec $t' < t$.

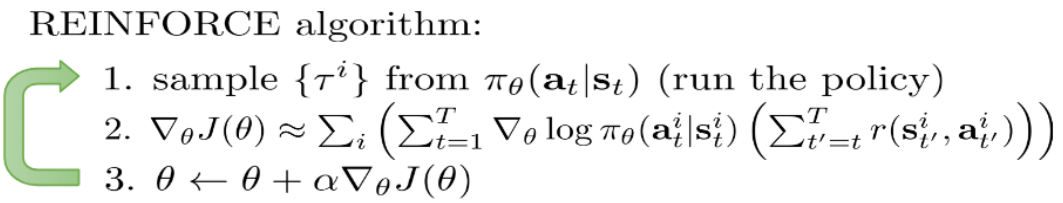


FIGURE 2.5 – Algo REINFORCE [Williams, 1992] avec la causalité, le t' part de t et non plus de 0. Image provenant de Levine [2017].

Il est important de noter que cet algorithme est *on-policy*. Cela veut dire qu'il faut ré-échantillonner de nouvelles trajectoires à chaque fois que l'on modifie la politique, i.e. que l'on effectue un pas de gradient. Cela peut donc être problématique si les expériences sont coûteuses à récolter car chaque expérience ne peut être utilisée qu'une seule fois.

2.3.2.2 Réduction de la variance du gradient de la politique grâce à l'ajout d'une référence

Il y a cependant différents problèmes dans l'algorithme REINFORCE lui-même. En effet, même s'il possède des propriétés de convergence théorique car c'est un algorithme du gradient stochastique, il peut être extrêmement instable et ne jamais converger en pratique. Cela est dû à une grande variance qui peut être réduite grâce à deux améliorations : l'utilisation d'une *référence* (*baseline* en anglais) tout d'abord, et enfin l'utilisation d'une architecture acteur-critique que nous décrirons plus en détail ci-dessous.

Le premier problème est que l'algorithme REINFORCE simple dépend de la valeur absolue des récompenses elles-mêmes. Par exemple, si on rajoute artificiellement une récompense énorme et constante à chaque pas de temps, la solution optimale reste inchangée (en faisant l'hypothèse que les épisodes sont de durées constantes) or cela peut rendre l'entraînement beaucoup plus instable. En réalité, seule l'information relative est pertinente : on souhaite savoir si une trajectoire est meilleure qu'une autre plutôt qu'obtenir une valeur absolue arbitraire.

En fait, on peut généraliser le théorème du gradient de la politique en ajoutant une référence $b(s_t)$ qui doit être indépendante de l'action prise (on peut démontrer que l'espérance du gradient reste inchangée) i.e. :

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t))(\sum_{t'=t}^{T} \gamma^{(t'-t)} r(s_{t'}, a_{t'}) - b(s_t))] \tag{2.18}$$

Une référence très souvent utilisée en pratique est une estimation de la fonction de valeur V_{π_θ}. L'idée intuitive est d'enlever la moyenne des récompenses possibles en partant de cet état pour atteindre une meilleure stabilité. En pratique, la fonction de valeur est estimée en utilisant les méthodes introduites dans la section précédente, algorithmes *fondés sur la fonction de valeur* 2.3.1.

2.3.2.3 Architecture acteur-critique

Malgré l'ajout d'une référence, l'algorithme reste instable et difficile à faire converger en pratique. Cette instabilité provient de la variance inhérente aux estimateurs de Monte Carlo ($Q(s_t, a_t) \approx \sum_{t'=t}^{T} \gamma^{(t'-t)} r(s_{t'}, a_{t'})$). En effet, la politique et même l'environnement peuvent être stochastiques, les trajectoires et donc les récompenses peuvent grandement varier d'un échantillon à l'autre : on fait une estimation avec un seul échantillon d'où une très grande variance. Pour illustrer cette idée, prenons l'exemple d'une voiture qui doit s'insérer dans un rond-point. Si la politique actuelle consiste à accélérer quoi qu'il arrive et que par chance il n'y a pas de voiture lors de l'évaluation de la trajectoire, on va accorder une très bonne récompense à cette trajectoire, car la voiture aura traversé le rond-point particulièrement vite sans aucune collision. L'idéal serait en fait d'estimer la valeur de chaque état en faisant de multiples essais et en moyennant les récompenses obtenues. On aurait ainsi pu constater que dans de nombreux cas, la stratégie actuelle amenait à des collisions et qu'il faut parfois freiner. Mais ce processus est rapidement de complexité trop importante et il est même parfois impossible car il n'est pas forcément faisable de revenir en arrière à un état précis.

L'idée de l'architecture acteur-critique est de ne plus utiliser les estimateurs de Monte Carlo pour estimer la Q-fonction mais de faire du boostrapping comme pour les méthodes TD. L'intuition est que la fonction de valeur apprend intrinsèquement une moyenne des récompenses possibles ce qui permet de réduire la variance du gradient de la politique comparé à la variance que l'on obtient en prenant un échantillon d'une trajectoire entière. En contrepartie, cela rajoute du biais et on perd la propriété de convergence théorique, car en boostrappant on ajoute une erreur de prédiction. Cependant, on utilise quasiment toujours en pratique des architectures acteur-critique lorsque l'on applique des algorithmes de RL fondés sur la politique. L'architecture acteur-critique est en fait une combinaison du théorème de gradient de politique avec les algorithmes TD fondés sur la fonction de valeur.

$$\text{On a donc } \sum_{t'=t}^{T} \gamma^{(t'-t)} r(s_{t'}, a_{t'}) \approx Q(s_t, a_t) \approx r_{t+1} + \gamma V_{\pi_\theta}(s_{t+1}) \tag{2.19}$$

Pour résumer, l'estimation du gradient (qui est maintenant biaisée) devient :

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t))(r(s_t, a_t) + \gamma \hat{V}_{\pi_\theta}(s_{t+1}) - \hat{V}_{\pi_\theta}(s_t))] \tag{2.20}$$

Il faut donc maintenir une estimation de la politique $\hat{\pi}_\theta$ (acteur) ainsi que de la fonction de valeur $\hat{V}_{\pi_\theta}$ (critique). Le choix des actions ne passe pas par un *argmax* sur l'ensemble des actions et peut donc s'appliquer à un ensemble continu d'actions, contrairement à l'algorithme du Q-Learning.

2.3.3 Fondés sur un modèle de l'environnement

Dans les parties précédentes, nous avons fait l'hypothèse que le modèle de l'environnement, i.e. la fonction de transition et la fonction de récompense, n'était pas connu. Dans cette partie, nous verrons que si le modèle est connu, alors on peut utiliser des algorithmes de *planification*, c'est à dire des algorithmes qui vont se projeter dans le futur afin de choisir l'action menant à la meilleure trajectoire à long terme. Dans le cas qui nous intéresse (la voiture autonome) le modèle est inconnu. En effet, nous n'avons pas accès aux intentions des autres usagers de la route (véhicules et piétons), ni aux changements d'états des feux tricolores ni même à l'évolution générale des conditions de luminosité et de météo. L'environnement de la voiture autonome semble donc très difficile à modéliser et n'est absolument pas un environnement contrôlé, dans le sens où il ne dépend pas seulement des actions prises par l'agent. Nous n'avons par conséquent qu'assez peu travaillé avec ce type d'algorithmes au cours de cette thèse et nous exposerons donc moins cette famille.

En introduction, nous avons présenté AlphaGo [Silver et al., 2016] et AlphaGo Zero [Silver et al., 2017] comme l'un des plus grands succès du DRL qui a été particulièrement médiatisé. Dans le cas du jeu de Go, on sait parfaitement quelle est la fonction de transition et quelle est la récompense que l'on va obtenir (gagner ou perdre), ces fonctions sont parfaitement définies par les règles du jeu. Par exemple, on sait précisément si un coup est légal ou non, ou bien quel sera l'état du plateau après avoir posé une pierre. Dans le cas où les actions sont discrètes (comme pour le Go ou les échecs), on peut ainsi en théorie créer un arbre de toutes les possibilités. Mais cet arbre croit exponentiellement avec le nombre d'actions possibles et il est donc totalement insoluble en pratique pour les jeux comme les échecs et surtout pour le Go.

AlphaGo est en fait fondé sur un algorithme nommé *Monte Carlo Tree Search* (MCTS) [Coulom, 2006], qui estime l'action optimale en effectuant des *rollouts*. L'idée d'un rollout est d'estimer la valeur d'un état futur en effectuant de nombreuses simulations partant de cet état jusqu'à la fin de l'épisode comme illustré Figure 2.6. Pour

pouvoir effectuer un grand nombre de simulations, les actions doivent être prises selon une politique extrêmement rapide, comme par exemple une politique aléatoire. On garde ensuite en mémoire le nombre d'évaluations ainsi que le nombre de victoires pour obtenir le pourcentage de victoire pour chaque état ainsi évalué. L'algorithme le plus simple est ensuite de juste prendre l'action menant au plus grand pourcentage de victoire.

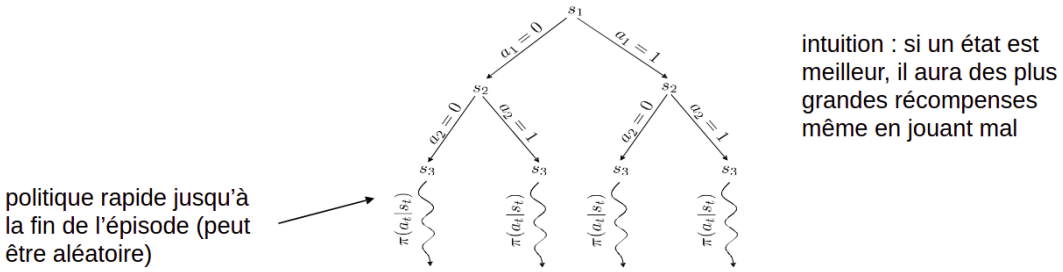


FIGURE 2.6 – Illustration de l'évaluation d'un état par *rollout*. Image provenant de Levine [2017].

L'algorithme MCTS est plus subtil que cela et effectue plus de rollouts depuis les états les plus prometteurs et construit ensuite un arbre plus intelligemment que juste en étendant de manière identique chaque branche de l'arbre, cela est plus détaillé dans l'article de R. Coulom [Coulom, 2006]. Pour résumer, l'intuition de cet algorithme est d'évaluer les états les plus prometteurs (exploitation) ainsi que ceux les moins visités (exploration), on retrouve ainsi le compromis exploration-exploitation si important dans l'apprentissage par renforcement.

Dans le cas où les actions sont continues, il existe une solution optimale analytique si la fonction de transition est linéaire et la fonction de récompense est quadratique sur la paire état-action (problème de contrôle LQG [Vinter, 2000] pour *Linear–Quadratic–Gaussian*). Ce problème est un des problèmes fondamentaux du contrôle optimal [Vinter, 2000]. Dans la majorité des cas, ces contraintes ne sont pas respectées et on utilise une expansion de Taylor pour linéariser la fonction de transition localement et quadratiser la fonction de récompense. Cela permet d'améliorer itérativement la trajectoire, plus de détails peuvent être trouvés dans l'article de E. Todorov [Todorov and Weiwei Li, 2005].

Même si le modèle de l'environnement n'est pas connu, on peut essayer de l'approximer par apprentissage supervisé. En utilisant des expériences provenant d'une politique quelconque, on veut prédire s_{t+1} et r_{t+1} à partir de s_t et a_t. L'idée est d'ensuite utiliser ce modèle pour effectuer des méthodes de planification comme décrit précédemment. Par contre, le modèle n'étant qu'une approximation, les performances finales peuvent en être grandement affectées. En outre, ce procédé permet d'apprendre des informations sur le monde dans lequel l'agent interagit sans nécessité de vérité terrain extérieure. En particulier certains travaux [Jaderberg et al., 2016, Shelhamer et al., 2016] utilisent ce principe pour ajouter des termes auxiliaires à la fonction de coût, l'intuition étant que des caractéristiques utiles pour prédire la récompense instantanée et l'état suivant peuvent aider à trouver les actions optimales.

2.4 Apprentissage par renforcement et réseau de neurones

Dans la partie 2.3.1 sur la famille fondée sur la fonction de valeur, nous avons admis qu'il était possible de maintenir une représentation tabulaire parfaite de V_π. Cependant, dans la plupart des applications concrètes, l'espace d'états est si grand qu'il est illusoire d'espérer pouvoir maintenir une représentation tabulaire de la fonction de valeur. Dans ces cas, on peut utiliser des fonctions paramétrées par des paramètres θ pour approximer cette fonction de valeur, on a donc $\hat{V}(s_t, \theta) \approx V_{\pi^*}(s_t)$. De la même manière, pour les algorithmes fondés sur la politique, on cherche à approximer la politique optimale, i.e. $\hat{\pi}(s_t, a_t, \theta) \approx \pi^*(s_t, a_t)$.

Les réseaux de neurones sont une excellente méthode permettant d'approximer des fonctions complexes à partir d'une grande quantité de données. Ils ont permis dernièrement de grandement améliorer les performances dans de nombreuses applications d'IA. Ils ont aussi été largement utilisés par tous les succès pratiques récents d'application de l'apprentissage par renforcement, que ce soit pour le jeu de Go [Silver et al., 2016, 2017], les jeux vidéos [Mnih et al., 2015, Vinyals et al., 2019] ou bien la robotique [Akkaya et al., 2019].

Traiter de manière approfondie les différentes manières de faire du RL avec des fonctions d'approximation est bien au delà de la portée de cet état de l'art mais pour résumer rapidement, à part dans de rares cas d'approximations à partir de fonctions linéaires, toutes les convergences théoriques des différents algorithmes sont perdues. Les réseaux de neurones étant par essence non linéaires, cela signifie qu'aucun algorithme de renforcement s'appuyant sur des réseaux de neurones n'a des propriétés de convergence théorique : on rappelle que dans le cas tabulaire SARSA et Q-Learning (section 2.3.1.3) convergent vers la politique optimale. Bien que les convergences ne soient absolument pas garanties, ces algorithmes s'inspirent des idées du cas tabulaire et grâce à cela l'apprentissage par renforcement profond a connu de grands succès pratiques récemment.

Par contre, ces algorithmes sont de manière générale très instables et de nombreuses astuces sont nécessaires pour leur permettre de converger en pratique. Cela rend aussi la réplicabilité des résultats très complexe car le moindre changement peut complètement annihiler l'apprentissage : même la graine du générateur des nombres aléatoires peut avoir un impact notable... Ces problèmes inhérents à l'apprentissage par renforcement en utilisant des réseaux de neurones comme fonctions d'approximation seront plus particulièrement détaillés dans le Chapitre 4.

Dans tout le reste de cette thèse, nous parlerons toujours d'apprentissage par renforcement utilisant des réseaux de neurones comme fonctions d'approximation (DRL pour Deep Reinforcement Learning).

2.4.1 Deep Q Network (DQN) : le "premier" algorithme d'apprentissage par renforcement profond

L'un des premiers succès du DRL et qui a contribué très fortement à l'engouement récent pour ce domaine a été DQN [Mnih et al., 2015], un algorithme qui a appris à jouer à 49 jeux Atari différents en utilisant comme seules informations les pixels de

l'écran de jeu et le score (pour la récompense). Il est particulièrement notable que la même architecture (même réseau de neurones et mêmes hyperparamètres) a appris à jouer à tous les jeux et certains avec une meilleure performance que celle d'un humain "moyen". Cependant, les performances de DQN restent très éloignées des meilleurs scores humain réalisés, comme nous le verrons dans le Chapitre 4.

Cet algorithme est une extension très simple de l'algorithme Q-Learning qui utilise un réseau de neurones pour approximer la Q-fonction optimale, $Q(s, a, \theta) \approx Q^*(s, a)$. Le réseau de neurones est optimisé par rétropropagation en utilisant la fonction de coût suivante :

$$L(s_t, a_t, r_{t+1}, s_{t+1}, \theta) = (\underbrace{r_{t+1} + \gamma \max_a Q(s_{t+1}, a, \theta)}_{cible} - Q(s_t, a_t, \theta))^2 \tag{2.21}$$

On reconnaît l'équation de mise à jour de l'algorithme du Q-Learning. La première partie de la fonction de coût (la *cible*) est considérée comme une constante ne dépendant pas de θ ce qui est faux en théorie, c'est pour cela que cette méthode s'appelle *semi-gradient* car ce n'est pas une véritable descente de gradient. L'intuition derrière cette fonction de coût est la suivante : si pour toutes les transitions $s_t, a_t, r_{t+1}, s_{t+1}$ le coût est nul, alors on sait qu'on a atteint la Q-fonction optimale. À noter que cela n'est pas une démonstration de la convergence de l'algorithme et on peut même trouver des cas où cet algorithme diverge. En fait, différentes astuces ont dû être utilisées afin de réellement rendre l'algorithme utilisable en pratique.

Le premier problème est que les expériences utilisées pour faire cette rétropropagation peuvent être extrêmement corrélées, par exemple deux images consécutives sur Atari seront extrêmement similaires. Pour résoudre cela, les chercheurs ont utilisés une *mémoire de reprise* (*replay memory* en anglais) dans laquelle les transitions $s_t, a_t, r_{t+1}, s_{t+1}$ sont stockées au fur et à mesure. Pour faire la rétropropagation avec des expériences approximativement décorrélées, on pioche des transitions aléatoirement dans cette mémoire. Cela est possible car l'algorithme du Q-Learning est un algorithme *off-policy*, on peut donc utiliser des échantillons provenant d'une autre politique (une version ancienne du réseau de neurones) pour faire la mise à jour des paramètres.

Le deuxième problème est que l'apprentissage avec cette fonction de coût est extrêmement instable et a tendance à osciller en pratique. Pour résoudre ce deuxième problème, Mnih et al. [2015] utilisent un *réseau cible* (*target network* en anglais) paramétré par θ' qu'ils utilisent pour calculer $\max_a Q(s_{t+1}, a, \theta')$. En pratique, le réseau cible est une copie figée du premier réseau de neurones et qui est remis à jour à des intervalles très espacés.

Un dernier problème peut aussi survenir lorsque les gradients sont extrêmement grands pouvant ainsi détruire tout l'apprentissage précédent. Pour résoudre cela, Mnih et al. [2015] limitent les gradients afin qu'ils restent dans des ordres de grandeurs identiques tout au long de l'apprentissage.

Il est important de noter qu'aucune de ces améliorations n'a de véritables propriétés théoriques (l'algorithme n'est pas censé plus converger en théorie) mais qu'elles sont en

fait extrêmement importantes pour lui permettre de converger en pratique. Sans ces trois astuces, DQN n'apprend rien sur la plupart des jeux Atari, comme montré dans l'article de Nature sur DQN [Mnih et al., 2015].

Depuis, de nombreuses améliorations de cet algorithme ont été proposées, on peut citer en particulier n-step DQN, Double DQN [Van Hasselt et al., 2016], Prioritized Experience Replay [Schaul et al., 2015], Noisy Networks [Fortunato et al., 2017], Distributionnal DRL [Bellemare et al., 2017]... L'algorithme Rainbow [Hessel et al., 2018] qui est actuellement l'état de l'art pour les algorithmes fondés sur la fonction de valeur est une combinaison de nombreuses améliorations de DQN en un seul et même algorithme. Nous détaillerons ces différentes améliorations dans le Chapitre 4 car ils sont à la base de l'algorithme utilisé majoritairement au cours de cette thèse, Rainbow-IQN ApeX [Toromanoff et al., 2019].

2.4.2 Asynchronous Advantage Actor-Critic (A3C) : le "premier" algorithme acteur-critique de DRL

Quelques temps après DQN, la même équipe de chercheurs a publié un nouvel algorithme : Asynchronous Advantage Actor-Critic (A3C) [Mnih et al., 2016]. A3C est l'un des premiers algorithmes d'acteur-critique utilisant des réseaux de neurones comme fonction d'approximation. Il est essentiellement fondé sur la théorie évoquée en partie 2.3.2.3, et il est présenté sur la Figure 2.7.

1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it on the robot) ⇒ This in On-policy!
2. fit $\hat{V}^\pi_\phi(\mathbf{s})$ to sampled reward sums
3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \hat{V}^\pi_\phi(\mathbf{s}'_i) - \hat{V}^\pi_\phi(\mathbf{s}_i)$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i)\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
5. $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

FIGURE 2.7 – Algorithme de A3C [Mnih et al., 2016]. Image provenant de Levine [2017].

On définit l'avantage comme $A_\pi(s_t, a_t) = Q_\pi(s_t, a_t) - V_\pi(s_t)$. L'intuition derrière cette définition est de déterminer l'avantage à prendre une action qui n'est pas forcément celle qu'aurait pris la politique actuelle, comparé à juste suivre cette politique. Comme vu précédemment, on peut estimer l'avantage seulement à partir de V_π et ainsi calculer le gradient de la politique :

$$A_\pi(s_t, a_t) = Q_\pi(s_t, a_t) - V_\pi(s_t) \approx r_t + \gamma V_\pi(s_{t+1}) - V_\pi(s_t) \tag{2.22}$$

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t))\hat{A}_{\pi_\theta}(s_t, a_t)] \tag{2.23}$$

Cet algorithme maintient donc deux réseaux de neurones, π_θ paramétré par θ et $V^\pi_{\theta_v}$ paramétré par θ_v. En pratique les deux réseaux partagent la majorité de leur poids,

l'intuition étant que les caractéristiques permettant d'estimer la valeur sont très proches de celles nécessaires pour estimer l'action optimale.

Comme pour DQN, les chercheurs ont trouvé diverses astuces permettant à l'algorithme de converger en pratique. Tout d'abord, ils utilisent plusieurs agents en parallèle, ce qui permet de décorréler les états successifs tout en gardant le caractère *on-policy* de l'architecture acteur-critique. En effet, l'utilisation d'une mémoire de reprise comme pour DQN n'est possible que si l'algorithme est *off-policy*. Ensuite, Mnih et al. [2016] ont trouvé que les performances étaient meilleures en rajoutant l'entropie de la politique dans la fonction de coût. L'intuition derrière cette idée est d'améliorer l'exploration de l'agent en l'incitant à maintenir plusieurs actions probables. Finalement, ils utilisent un boostrap à *n-step* (comme pour le *n-step* TD décrit dans la partie 2.3.1.3) pour l'évaluation de l'avantage :

$$\hat{A}_\pi(s_t, a_t) \approx \sum_{k=0}^{t_{max}} \gamma^k r_{t+k} + \gamma^{t_{max}} \hat{V}_\pi(s_{t+t_{max}}) - \hat{V}_\pi(s_t) \text{ avec } t_{max} = 5 \tag{2.24}$$

A3C est l'algorithme que nous avons utilisé lors d'un travail préliminaire à cette thèse [Jaritz et al., 2018] que nous détaillerons dans le Chapitre 3.

2.5 Comment appliquer du DRL à la voiture autonome ?

Nous avons pour l'instant présenté les bases de la théorie de l'apprentissage par renforcement ainsi que les grandes familles d'algorithmes de renforcement. Nous allons maintenant nous intéresser spécifiquement au cas d'application de cette thèse : l'apprentissage du contrôle pour la voiture autonome. Nous allons d'abord voir quel environnement choisir et en quoi les simulateurs sont nécessaires pour l'apprentissage du contrôle d'un véhicule, particulièrement pour l'apprentissage par renforcement. Ensuite, nous discuterons des choix possibles pour la représentation de l'état et des actions, notamment des avantages et inconvénients d'une représentation *haut niveau* comparé à un apprentissage de *bout-en-bout*, i.e. directement depuis les données brutes des capteurs. Finalement, nous expliciterons les différences pratiques entre les différentes familles d'algorithme de renforcement pour choisir celle qui semble la plus adaptée à notre cas d'application.

2.5.1 Choisir l'environnement : les simulateurs pour la voiture autonome

Depuis l'explosion récente de l'utilisation des réseaux de neurones dans une grande variété d'applications de l'intelligence artificielle dont bien sûr la conduite autonome, le besoin en données a lui aussi explosé. En effet, les réseaux de neurones nécessitent une grande quantité de données pour converger et cela a amené divers laboratoires et entreprises à créer des simulateurs afin de générer automatiquement des bases de données voire même de faire des apprentissages directement dans la simulation. C'est pourquoi il y a eu récemment une apparition de nombreux acteurs spécialisés dans la

simulation pour la voiture autonome spécifiquement pour l'intelligence artificielle et les réseaux de neurones et non plus seulement pour faire de la validation comme cela était le cas auparavant. Mais ce domaine étant extrêmement récent, la grande majorité de ces acteurs, qu'ils soient open-source ou commerciaux, ne sont pas encore tout à fait matures.

De plus, l'apprentissage par renforcement repose fondamentalement sur la méthode essai-erreur, ce qui signifie que les cas d'échecs doivent être réellement rencontrés pour apprendre à les éviter. Malheureusement, cela n'est pas possible dans la majorité des applications réelles pour des raisons de sécurité et en particulier pour les voitures autonomes. C'est pourquoi l'utilisation d'un simulateur d'un environnement routier et urbain semble obligatoire pour espérer effectuer du renforcement pour la voiture autonome.

Dans l'idéal, une simulation pour la voiture autonome devrait répondre à différents critères que nous allons énumérer juste après. Comme nous le verrons, il n'existe pas vraiment de simulateur répondant à toutes ces conditions. Nous allons ici présenter rapidement différents environnements de simulation pour la voiture autonome en énumérant leurs points faibles et leur qualités, pour finalement expliquer pourquoi nous avons choisi CARLA [Dosovitskiy et al., 2017] comme simulateur privilégié.

Les critères que doit idéalement vérifier le simulateur (par ordre d'importance) :

— flexibilité de l'environnement : avoir directement accès au code de la simulation est obligatoire pour pouvoir développer, le plus prometteur étant d'utiliser des simulations utilisant un moteur de jeu vidéo comme Unity ou bien Unreal Engine

— variété de l'environnement (différentes conditions météo, luminosités, décors, végétations et bâtiments variés...). Nous allons en effet voir qu'il vaut mieux privilégier un environnement très varié à un environnement *photo-réaliste*.

— complexité et réalisme de l'environnement urbain (présence de feux tricolores, de panneaux de signalisation, minimum de réalisme graphique)

— présence d'objets mobiles (piétons, voitures, vélos...) se *comportant de manière réaliste* : ce point n'est pour l'instant vérifié par aucun simulateur

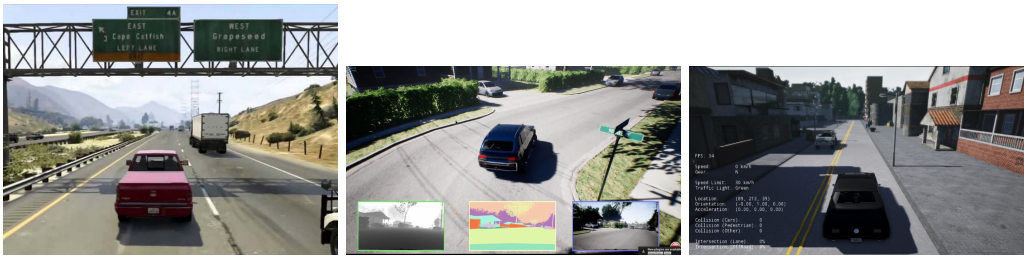
— vitesse d'exécution de la simulation suffisamment rapide pour pouvoir générer une quantité importante de données

— possibilité d'entraîner plusieurs agents en parallèle dans une même simulation (multi-agents)

On peut citer notamment CVEDIA , Righthook ou bien Cognata [1] qui vendent leur outil de simulation pour la conduite autonome spécifiquement dans le but d'entraîner des réseaux de neurones. Malheureusement, ces simulateurs nécessitent des licences et n'ont pas vraiment de version à tester, ils n'étaient donc pas accessibles dans le cadre de cette thèse.

Du coté des simulateurs open-source, cf Figure 2.8, on retrouve le jeu vidéo Grand Theft Auto (GTA) qui a été détourné pour permettre aux chercheurs ([Richter et al., 2016, Martinez et al., 2017] par exemple) de l'utiliser comme simulateur pour la conduite

1. www.cvedia.com, www.cognata.com, righthook.io



(a) GTA/DeepDrive.io (b) AirSim (c) CARLA

FIGURE 2.8 – Exemples d'images de différents simulateurs (les trois utilisent Unreal Engine, qui semble le moteur de jeu vidéo le plus approprié pour ce type d'application)

autonome. Ce jeu est grandement réaliste, les comportements des autres usagers sont particulièrement intéressants, le monde est très grand et très varié. Par contre le jeu est extrêmement lourd : il faut un ordinateur très puissant pour pouvoir exécuter ce jeu et il est presque impossible d'aller plus vite que le temps réel. Mais la véritable contrainte est la non-flexibilité, en effet ce jeu n'a pas du tout été développé dans l'intention d'être un simulateur, les moindres changements peuvent ainsi être extrêmement complexes à développer, les mises à jour (obligatoires) du jeu peuvent même parfois complètement casser le code précédemment utilisé. C'est d'ailleurs pour cela qu'une start-up, *DeepDrive.io*, est en train de redévelopper un simulateur très proche de GTA mais avec Unreal Engine, ce qui permettrait ainsi d'être flexible et d'une certaine manière de ne pas *hacker* le jeu pour effectuer un changement dans la simulation. Malheureusement, ce projet n'était pas encore assez abouti pour être utilisé dans le cadre de cette thèse.

Microsoft a récemment mis en libre service son simulateur fondé lui aussi sur Unreal Engine, d'un environnement urbain *AirSim* [Shah et al., 2018] initialement développé pour les drones. Ils ont aussi rapidement rajouté une fonctionnalité pour intégrer des voitures. Cependant, l'environnement est assez petit et ne possède pas réellement d'objets mobiles.

Simulateur	GTA	DeepDrive.io	AirSim	CARLA
Flexibilité	– –	++	++	++
Variété	++	– –	–	+
Complexité/Réalisme	++	– –	–	–
Objets mobiles	++	– –	– –	+
Vitesse éxecution	– –	+	+	+
Multi-agent	– –	–	–	++

TABLE 2.1 – Comparaison des différents simulateurs open-source disponibles. DeepDrive.io a pour objectif de reproduire GTA sur Unreal Engine mais n'est pour l'instant qu'au stade initial de développement.

Finalement, le simulateur open-source le plus mature aujourd'hui (voir Table 2.1) est

le simulateur de l'université de Barcelone, *CARLA* [Dosovitskiy et al., 2017]. Ce simulateur possède plusieurs environnements urbains, permet de rajouter automatiquement des piétons et d'autres véhicules même si leurs comportements restent pour l'instant assez simples. De plus, ils présentent un benchmark spécifiquement destiné à évaluer des algorithmes d'apprentissage de contrôle dans leur simulateur. Ce simulateur a donc été développé dans l'optique de pouvoir appliquer du renforcement. Finalement, il est aussi possible de lancer plusieurs agents en parallèle dans la même simulation, ce qui permet de générer plus facilement un grand nombre d'expériences. C'est pour ces raisons que CARLA a été le simulateur le plus utilisé pour les travaux en simulation de cette thèse. Nous avons ainsi pu participer au CARLA challenge [2] que nous détaillerons dans le Chapitre 5.

2.5.1.1 Transfert d'apprentissage de la simulation au monde réel

Comme décrit précédemment, le RL nécessite d'apprendre sur un simulateur dans la grande majorité des cas. Pour certaines tâches, e.g. les échecs, le go, les jeux vidéos, il n'y a pas de différence entre le simulateur et le cas d'application concret et il n'y a donc pas besoin de faire de transfert d'apprentissage. Par contre, pour les cas d'applications *réels*, il peut y avoir un fossé (*domain gap*) entre la simulation et la réalité. C'est pourquoi le transfert d'une simulation au monde réel est l'un des thèmes majeurs en apprentissage par renforcement.

Le principal problème du transfert d'apprentissage est que l'algorithme de RL va optimiser exactement la tâche sur laquelle il apprend, i.e. la simulation. Il va s'appuyer sur quoi que ce soit qui puisse l'aider à augmenter ses récompenses, même si cela est très loin du véritable but voulu.

L'intuition la plus simple pour améliorer le transfert d'une simulation au monde réel est de se dire que plus la simulation est réaliste et proche du monde réel, plus le transfert sera simple. Grâce au progrès des environnements virtuels, en particulier dans le domaine des jeux vidéos, il est en effet possible d'obtenir des effets (en particulier graphiques) impressionnants et de plus en plus réalistes. Le problème étant que ces effets graphiques sont généralement extrêmement coûteux : on perd ainsi l'un des intérêts d'une simulation qui était de pouvoir s'exécuter extrêmement rapidement.

Une autre idée intéressante serait d'entraîner (dans la simulation) et de tester (dans le monde réel) dans le même espace de représentation. On peut penser à une image segmentée sémantiquement ou à une représentation avec la position et l'orientation de tous les objets de la scène par exemple. Mais cela induit une dépendance à la performance de l'algorithme de perception qui ferait cette segmentation ou cette détection d'objets lors du test dans le monde réel (dans la simulation, on peut avoir accès à ces informations de manière parfaite).

Ainsi, dans l'article Virtual to Real Reinforcement Learning for Autonomous Driving [Pan et al., 2017], les chercheurs ont pour idée d'utiliser un GAN (pour Generative

2. https ://carlachallenge.org

Adversarial Network [Goodfellow et al., 2014]) conditionnel [Isola et al., 2017] pour générer des images réelles à partir d'images segmentées issues de la simulation. L'idée étant d'entraîner ce GAN à partir de datasets avec des images réelles segmentées, et de le tester sur les images segmentées issues de la simulation. On pourrait donc en théorie apprendre dans la simulation avec ces images générées *réelles* pour ainsi tester dans le monde réel sans avoir besoin de faire du transfert. Mais pour l'instant la performance des GAN n'est pas encore suffisante pour cela, et il reste encore des progrès à faire pour espérer faire fonctionner cette idée dans le monde réel. Dans leur article, Pan et al. [2017] ont étudié le transfert d'apprentissage d'un simulateur à un autre mais pas sur une vraie voiture.

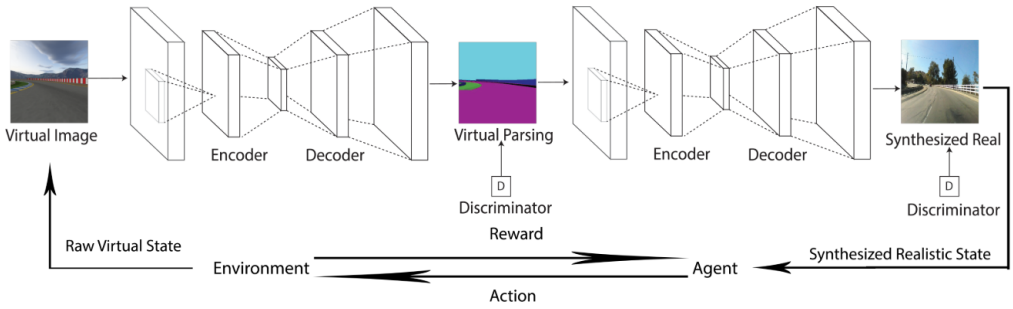


FIGURE 2.9 – Schéma du principe de l'article de Pan et al. [2017] pour effectuer du renforcement sur des images réelles. L'idée est d'abord de passer dans une représentation intermédiaire (ici la segmentation) puis d'utiliser un GAN conditionnel pour générer une image réelle représentant la même scène. Cette image finale est ainsi l'état courant pour l'agent de renforcement. Image provenant de Pan et al. [2017].

Finalement, une autre idée pour effectuer le transfert d'une simulation au monde réel est la *randomisation du domaine*. Le principe est plutôt qu'essayer de faire une simulation la plus réaliste possible, essayer d'entraîner sur des environnements les plus différents possibles mais qui garderait toujours le même objectif. De plus, ces environnements n'ont pas nécessairement besoin d'être réalistes et peuvent souvent être générés facilement ce qui permet de créer un grand nombre d'expériences rapidement. Ce principe voit en fait le monde réel comme une énième simulation générée aléatoirement. Dans l'article de Tobin et al. [2017], les chercheurs ont ainsi réussi à transférer dans le monde réel un modèle entraîné uniquement en simulation. Cet agent devait contrôler un bras robotique réel pour s'emparer d'un objet d'une certaine forme.

Pour cela Tobin et al. [2017] ont fait varier aléatoirement de nombreux paramètres de la simulation à chaque épisode, comme le nombre, la forme, la couleur et la position des autres objets de la scène ainsi que la position, l'orientation et le champ de vision de la caméra. Finalement, ils modifient aussi le nombre de lumières et l'éclairage de la scène. Des exemples de ces différentes simulations aléatoires sont présentées sur la Figure 2.10.

FIGURE 2.10 – Exemple de randomisation de domaine dans le cadre d'une simulation d'un bras robotique : l'objectif est toujours de réussir à attraper l'objet de forme cylindrique, quelque soit sa couleur, l'éclairage, les autres objets ou la position de la caméra. Image provenant de Tobin et al. [2017].

2.5.1.2 Complications liées au transfert d'une simulation au monde réel pour la voiture autonome

Même si le transfert d'apprentissage est une piste extrêmement intéressante pour espérer faire de l'apprentissage par renforcement sur voiture réelle, il y a encore de nombreux obstacles et problèmes à résoudre.

Le problème fondamental est qu'il y aura toujours un fossé plus ou moins grand entre la simulation et le monde réel. Les techniques que nous avons vu précédemment s'intéressent principalement aux différences de réalisme entre les images de la simulation et du monde réel. Cependant, il existe au moins deux autres aspects qui peuvent être totalement différents entre la simulation et le monde réel et qui peuvent avoir un impact énorme sur la performance que l'on peut espérer atteindre sur le véhicule réel.

Premièrement, le comportement des autres objets (piétons, voitures, vélos...) est généralement extrêmement simple dans les simulations comparé aux comportements dans le monde réel. Ainsi, l'agent ne pourra jamais espérer apprendre des interactions complexes avec les autres usagers de la route car dans la simulation ces interactions seront extrêmement simplifiées. De même, si la simulation n'inclut pas de panneau de signalisation, ou certains types de marquages, l'agent ne pourra jamais apprendre comment y réagir. Dans ce cas, la seule chose qu'on pourrait espérer faire est de détecter les cas où l'on est en dehors de la distribution d'entraînement. Cependant ce sujet étant déjà

extrêmement complexe pour la perception pure, il n'est pas encore mature pour être appliqué jusqu'au contrôle.

Deuxièmement, la physique de la simulation (le contrôle du véhicule, les glissements sur le sol) est généralement très simplifiée dans les simulations. Par exemple dans CARLA, l'agent va apprendre qu'il peut freiner quasiment instantanément car le modèle d'inertie est très simplifié ce qui n'est bien évidemment pas le cas dans le monde réel. Finalement, il peut y avoir d'innombrables points qui ne sont pas présents dans la simulation : par exemple CARLA ne possède pas d'intersections en fourche, l'agent ne saura donc pas comment réagir dans ces cas.

En fait, certains articles récents ([Osiński et al., 2020, Amini et al., 2020]) ont appliqué des méthodes de transfert depuis le simulateur CARLA pour le contrôle du volant d'un véhicule réel. En particulier, Amini et al. [2020] ont utilisé des méthodes de *randomisation de domaine* illustrés Figure 2.11.

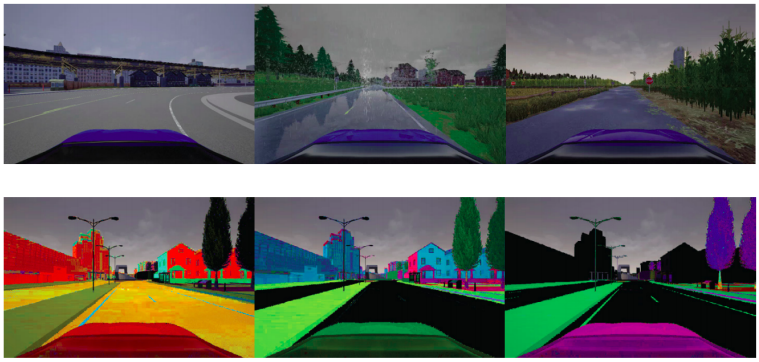


FIGURE 2.11 – Exemples de randomisation de domaine dans le simulateur CARLA proposés par Amini et al. [2020]. Le but est ensuite de transférer l'apprentissage en testant l'agent pour le contrôle du volant dans une voiture réelle. Image provenant de Amini et al. [2020].

Cependant, les résultats de ces deux travaux ([Osiński et al., 2020, Amini et al., 2020]) sont très mitigés et montrent bien que la randomisation de domaine n'est pas suffisante pour transférer un apprentissage d'un simulateur à un véhicule réel. Il est important de noter que l'article de Amini et al. [2020] est le premier travail à avoir réussi à faire fonctionner de l'apprentissage par renforcement sur une voiture réelle en apprenant dans un *simulateur utilisant des images réelles*. Nous reviendrons en détail sur cet article dans le Chapitre 7.

Tous ces points m'ont poussé à m'affranchir du transfert d'apprentissage car le transfert est en soi déjà une tâche extrêmement complexe et ne pourrait donc s'appliquer actuellement qu'à des tâches simples et pas à des situations plus complexes comme la conduite en environnement urbain. C'est pour cela que nous avons fait le choix dans cette thèse de ne pas nous intéresser au transfert d'apprentissage d'un environnement à l'autre. Pour nos cas d'applications, nous avons toujours appris dans l'environnement final mais le test s'effectue quand même dans des conditions différentes. Par exemple

pour le CARLA challenge, nous n'avions pas accès aux villes utilisées pour l'évaluation finale, ce qui implique malgré tout une grande capacité de généralisation. Pour effectuer de l'apprentissage par renforcement sur données réelles, nous avons construit un simulateur utilisant des images réelles que nous détaillerons dans les Chapitres 6 et 7.

2.5.2 Choisir la représentation de l'état et des actions : Apprentissage des capteurs au contrôle, ou représentation *haut niveau*

Un autre aspect fondamental dans l'application d'un apprentissage par renforcement profond est le choix de la représentation de l'état, i.e. sous quelles formes l'agent va recevoir les informations de l'environnement. Dans le cadre de la voiture autonome, nous allons voir que cette représentation peut être très diverse et va grandement diriger les cas d'applications concrets possibles (autoroute, maintien de voie, insertion dans un rond-point etc...). Nous allons d'abord parler des représentations *haut niveau* possibles et de différent travaux de renforcement s'appuyant sur ce type de représentations. Puis, nous introduirons les différents problèmes qui surviennent lorsque l'on passe par des représentation haut niveau. Finalement, nous expliciterons le choix final de cette thèse, i.e. un apprentissage de *bout-en-bout* (en anglais *end-to-end*) des capteurs au contrôle.

2.5.2.1 Représentation haut niveau de la perception

Une très bonne revue des différentes représentations haut niveau possibles de l'espace d'états et d'action pour la voiture autonome peut se retrouver dans l'article *A Survey of State-Action Representations for Autonomous Driving* [Leurent, 2018]. Pour résumer, une représentation simple d'un véhicule peut se définir par sa position (x, y) sur le sol (dans la grande majorité des cas, on va négliger les possibles variations d'altitude), par son orientation θ ainsi que sa vitesse actuelle v. On a $s_{veh} = [x\ y\ \theta\ v]$.

Ainsi, pour rajouter plusieurs véhicules on peut représenter l'état global s par le set composé des états individuels $s_{k\in[0,N]}$ de chaque véhicule comme illustré Figure 2.12.

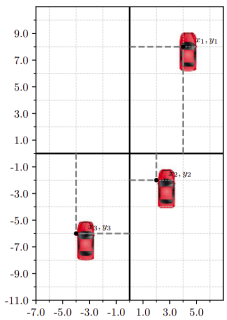


FIGURE 2.12 – Exemple de discrétisation des positions de plusieurs véhicules. Image provenant de Leurent [2018].

Une autre information très importante dans le cadre de la voiture autonome est la position des différentes voies de circulation ainsi que la position de chaque véhicule dans leur

voie. Il est ainsi assez courant de maintenir un système de coordonnées curvilignes pour chaque voie, pour fournir la position de chaque objet dans sa propre voie, cf Figure 2.13. Ce type de représentation est utilisé majoritairement dans les travaux appliquant de l'apprentissage par renforcement profond à la conduite sur autoroute [Mirchevska et al., 2018, Bai et al., 2019, Albaba and Yildiz, 2020]. Dans ces trois travaux, l'environnement est une autoroute multi-voies dans lequel l'agent doit optimiser sa vitesse tout en évitant les collisions. L'espace d'actions est lui aussi un espace d'actions *haut niveau*, c'est à dire des prises de décisions qui sont ensuite exécutées par des algorithmes de contrôle. Dans ces trois articles, les actions consistent à soit rester sur sa voie, soit changer de voie, puis la commande réelle est ensuite calculée en fonction de la position actuelle du véhicule et celle de la voie choisie par l'agent appris par renforcement. L'agent observe le monde via différentes informations haut niveau, la distance et la vitesse relative aux véhicules proches, sur quelle voie se trouve chaque véhicule. Dans ces trois cas, l'algorithme de renforcement lui-même est DQN [Mnih et al., 2015].

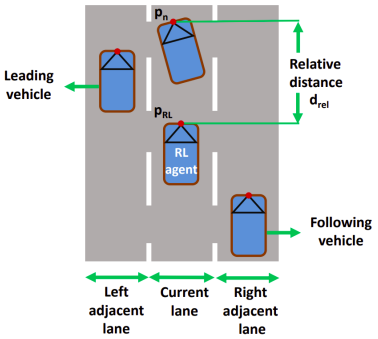


FIGURE 2.13 – Exemple de représentation pour entraîner un agent à conduire sur autoroute, l'idée est que l'agent peut effectuer des actions haut niveau pour soit changer de voie, soit garder sa voie actuelle. Image provenant de Mirchevska et al. [2018].

Finalement, d'autres articles rajoutent des paramètres afin de complexifier l'environnement et la tâche à résoudre. Dans [Kamran et al., 2020] et [García Cuenca et al., 2019], les chercheurs s'intéressent spécifiquement au cas d'insertion et de sortie d'intersection et de rond-point. Pour cela ils rajoutent une information sur la position relative des différentes sorties de l'intersection, ainsi qu'une commande pour indiquer à l'agent quelle sortie il doit prendre. Kamran et al. [2020] prennent aussi en compte la distance relative de chaque véhicule à l'intersection dans l'espace d'états comme illustré Figure 2.14. Dans le premier travail, Kamran et al. [2020] ne s'intéressent pas au contrôle de l'angle au volant et l'espace d'actions est relativement haut niveau composé de seulement 3 actions : s'arrêter, aller à une vitesse faible ou aller à grande vitesse. L'idée étant que la commande bas niveau sera effectuée via un algorithme de contrôle classique type PID. Dans le deuxième article [García Cuenca et al., 2019] par contre, les actions sont discrétisées mais bas niveau, i.e. l'agent va directement calculer une commande, un triplet (accel, angle, frein), qui va être réellement exécutée dans la simulation.

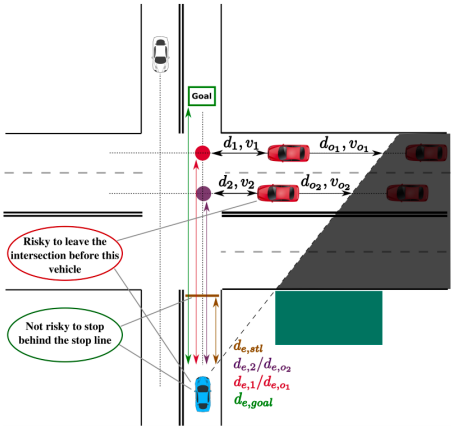


FIGURE 2.14 – Exemple de représentation pour entraîner un agent à s'insérer et à sortir d'une intersection avec la présence d'autres agents. L'agent peut effectuer des actions haut niveau pour soit s'arrêter, aller à faible vitesse ou forte vitesse. Les commandes bas-niveaux sont elles exécutées par un algorithme de contrôle classique type PID. Image provenant de Kamran et al. [2020].

2.5.2.2 Problèmes liés à la représentation haut niveau

Nous avons donc donné quelques exemples de représentation haut niveau pour l'espace d'états et l'espace d'actions, et expliqué brièvement quelques références de travaux utilisant ce type de représentation pour de l'apprentissage par renforcement profond. Malheureusement on peut voir que l'ensemble de ces travaux ne s'appliquent que sur une tâche très spécifique et bien loin de la tâche globale consistant à conduire une voiture autonome prenant en compte les intersections, les voies, les piétons, les feux, les panneaux etc... Il y a en fait plusieurs problèmes majeurs survenant lorsque l'on passe par une représentation haut niveau.

Le premier problème qui survient avec une représentation haut niveau est qu'il est très difficile de gérer des cas généraux. En effet, il faudrait potentiellement rajouter des milliers de paramètres différents. Pour illustrer avec des exemples simples, il faudrait potentiellement rajouter pour chaque véhicule son type (c'est à dire camion, vélo, remorque etc...) car cela peut avoir une conséquence sur son comportement. De même pour les piétons, connaître leur orientation, savoir s'ils nous ont vu ou non, sont autant de paramètres qui ont un impact sur la conduite à prendre. On voit qu'on est très vite limité et qu'il n'est pas concevable de pouvoir traiter l'ensemble des cas possibles via une représentation haut niveau.

Un deuxième problème majeur avec une représentation haut niveau est que l'on ne sait pas nécessairement quelles sont les informations les plus importantes pour la tâche voulue. On cherche en quelque sorte des *caractéristiques* (*features*) haut niveau pour permettre à l'algorithme d'apprendre plus facilement. On peut faire un parallèle avec les caractéristiques faites à la main pour le traitement d'images (ou pour la perception en général) qui sont maintenant devenues presque obsolètes par rapport à prendre directe-

ment l'image brute en entrée d'un réseau de neurones. En effet, c'est le réseau de neurones qui va apprendre lui-même les caractéristiques les plus utiles des données d'entrée pour la tâche voulue. Dans l'idéal, ce même raisonnement devrait s'appliquer à la planification et au contrôle, i.e. laisser l'agent trouver lui même ses propres caractéristiques pour résoudre la tâche finale de conduite.

Un troisième problème avec les représentations haut niveau est que l'on admet que la perception est parfaite. Or, lorsque l'on voudra appliquer du renforcement dans un cas réel, on n'aura pas accès à ces informations privilégiées et on devra se reposer sur des algorithmes de perception qui vont nécessairement produire des informations bruitées et même des erreurs. L'algorithme entraîné sur une représentation haut niveau ne sera probablement pas robuste à ce bruit ou ces erreurs. Il est par contre possible d'entraîner l'agent à partir d'informations provenant déjà de l'algorithme de perception final afin de rendre l'agent plus robuste aux erreurs (mais si la perception est trop bruitée, la tâche en sera d'autant plus complexe).

2.5.2.3 Apprentissage de bout-en-bout : des capteurs au contrôle

En fait, lorsque les humains conduisent, ils ne s'appuient que sur la vision binoculaire pour conduire (on pourrait rajouter le son mais qui semble à priori très secondaire) donc deux images doivent nécessairement contenir toutes les informations utiles à la conduite. On s'attend donc à ce qu'une IA soit capable d'égaler et même de surpasser l'humain avec seulement quelques images caméras en entrée (de plus les caméras sont des capteurs très peu coûteux). L'apprentissage de *bout-en-bout*, i.e. apprendre directement depuis les informations capteurs, semble donc le cas idéal dans le sens où l'on peut en théorie résoudre l'ensemble de la tâche globale de la conduite autonome avec une seule représentation commune. De plus, partir des capteurs permet de résoudre tous les problèmes cités précédemment qui surviennent lorsque l'on passe par une représentation haut niveau. Pour toutes ces raisons, nous avons fait le choix de faire de l'apprentissage de bout-en-bout pour tous les travaux de cette thèse.

Cependant, il y a un énorme obstacle à l'apprentissage de bout-en-bout : c'est extrêmement complexe, et c'est la raison principale pour laquelle on passe par des représentations intermédiaires et différents blocs perception/planification/contrôle. En effet, on utilise directement les données capteurs brutes (en particulier des images caméra), l'espace d'entrée est donc très grand ce qui rend l'apprentissage difficile. De plus, les tâches à résoudre sont très variées et les apprendre toutes en même temps est extrêmement compliqué. En fait, ce n'est que récemment qu'il y a eu des premiers succès de cas d'application d'apprentissage de bout-en-bout pour la conduite autonome. La grande majorité de ces travaux n'impliquent pas d'apprentissage par renforcement mais de l'apprentissage par imitation (en particulier par *behavorial cloning*) [Bojarski et al., 2016, Codevilla et al., 2018]. Nous ferons une description plus détaillée de ces travaux et de l'apprentissage par imitation plus généralement dans le Chapitre 6.

Les travaux de renforcement qui font de l'apprentissage de bout-en-bout pour la voiture autonome se sont d'abord principalement intéressés au cas des jeux de course comme un premier pas vers la tâche de la voiture autonome. Un de mes propres travaux [Jaritz

et al., 2018] rentre exactement dans cette catégorie, i.e. apprentissage par renforcement de bout-en-bout pour un jeu de course. J'ai effectué ce travail avant de commencer cette thèse et en ferai une brève description dans le Chapitre 3 tout en donnant un état de l'art du DRL appliqué aux jeux de courses.

Finalement, les travaux effectuant du DRL de bout-en-bout pour la voiture autonome en particulier pour la conduite en environnement urbain sont très récents et seront détaillés dans le Chapitre 5.

2.5.3 Quel algorithme de Renforcement choisir ?

Nous avons présenté de manière approfondie les différentes familles d'algorithmes de RL dans la section 2.3, la question est maintenant de déterminer quelle famille serait la plus apte à être utilisée dans le cas de la voiture autonome. Dans le tableau suivant, Table 2.2, nous résumons les points évoqués dans la partie précédente en rappelant les algorithmes et les caractéristiques principales de chacune de ses familles.

Famille	Algorithme	On/Off policy	Action continu/discret
Fondée sur la politique	REINFORCE	On policy	Les 2
	Acteur-critique	On policy	Les 2
Fondée sur la fonction de valeur	SARSA	On policy	Discret
	Q-Learning	Off-policy	Discret
Fondée sur un modèle	MCTS	Off-policy	Discret
	iLQG	Off-policy	Continu

Table 2.2 – Résumé des 3 familles d'algorithmes de renforcement et leurs caractéristiques principales.

On peut en fait approximativement classer ces algorithmes par *efficacité en données* (dans la littérature cela est référé sous le nom de *data efficiency*), i.e. ont-ils besoin de plus ou moins d'expériences pour converger, voir schéma Figure 2.15.

Même si ce schéma est évidemment très simplificateur, il donne une bonne intuition

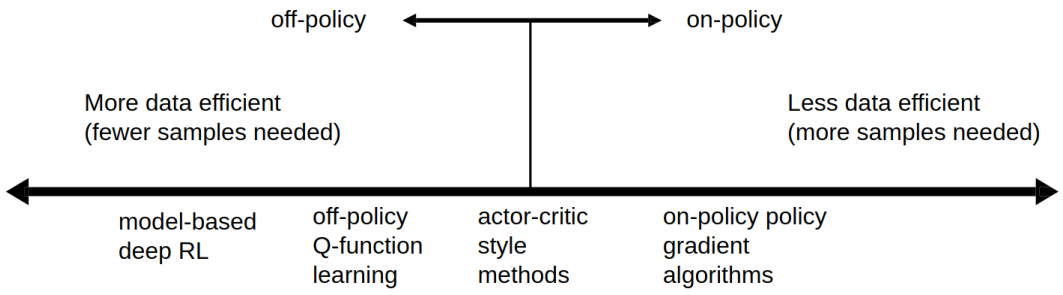


FIGURE 2.15 – Les différentes familles selon leur efficacité. Image provenant de Levine [2017]

des différentes familles. On retrouve bien le fait que les algorithmes on-policy sont généralement moins efficaces que les algorithmes off-policy car de nouveaux échantillons doivent être générés à chaque fois que l'on met à jour la politique. La question est donc pourquoi utiliser des algorithmes qui seraient moins efficaces ? Cela est en fait dû à plusieurs raisons, la plus simple étant que généralement les algorithmes fondés sur un modèle atteignent de moins bonnes performances finales que les autres, en particulier lorsque l'on doit passer par une approximation du modèle. Une autre raison est que dans certains cas, la simulation de nouvelles expériences est très peu coûteuse, i.e. peut aller bien plus vite que le temps réel (e.g. l'émulateur Atari), et dans ce cas le temps de l'apprentissage est généralement beaucoup plus long pour les algorithmes qui font de la planification. Inversement, si la génération d'expériences se fait dans le monde réel, il faut absolument essayer de retirer le plus d'informations possibles de chaque transition i.e. utiliser des algorithmes off-policy. Ces informations peuvent se résumer dans le graphe suivant, voir Figure 2.16.

D'après ce graphe, la famille qui semble la plus prometteuse est la famille fondée sur la valeur (DQN [Mnih et al., 2015], Rainbow [Hessel et al., 2018]) car comme nous l'avons décrit dans la partie 2.5.1, on aura bien un simulateur mais qui ne sera pas capable de s'exécuter beaucoup plus rapidement que le temps réel.

Un autre critère qui doit être noté est de savoir si l'algorithme est limité à des actions discrètes ou peut s'étendre à des actions continues. La majorité des algorithmes fondés sur la fonction de valeur (DQN [Mnih et al., 2015], IQN [Dabney et al., 2018]) ne permettent que des actions discrètes car il est nécessaire de trouver le maximum de la Q-fonction sur l'espace des actions dans l'algorithme du Q-Learning. Cependant, dans le cas de la voiture autonome l'espace des actions ne possède que 2 dimensions (l'angle au volant et l'accélération/freinage), et il semble possible à priori d'appliquer des algorithmes utilisant des actions discrètes pourvu que la discrétisation soit suffisamment fine.

Nous avons donc fait le choix d'utiliser des algorithmes *off-policy* fondés sur la fonction de valeur pour la majorité des travaux de cette thèse.

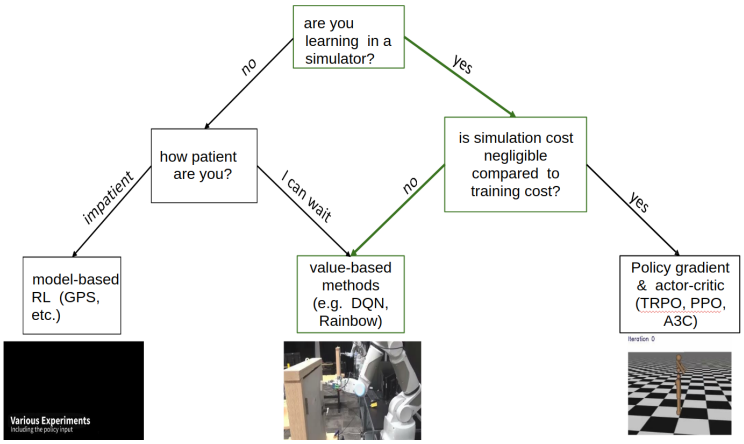


FIGURE 2.16 – Illustration très simplifiée pour choisir la famille d'algorithme la plus appropriée dans une certaine situation. Les flèches vertes correspondent au cas d'application de cette thèse, la voiture autonome. En effet, on se repose sur un simulateur d'un environnement urbain dont l'exécution a un coût de calcul qui est loin d'être négligeable. Image provenant de Levine [2017]

2.6 Conclusion : Choix techniques et d'application de la thèse

Nous avons d'abord explicité le choix de l'environnement et pourquoi le simulateur CARLA semblait le plus approprié pour appliquer de l'apprentissage par renforcement à la conduite en milieu urbain. Nous avons aussi expliqué que nous avions écarté l'hypothèse du transfert d'apprentissage d'un environnement à l'autre et nous expliquerons dans le dernier chapitre (Chapitre 7) comment nous avons procédé pour entraîner un algorithme de RL directement sur des images réelles. Puis, nous avons montré pourquoi nous nous sommes orientés vers de l'apprentissage de bout-en-bout pour pouvoir l'appliquer au cas complexe de la conduite urbaine. Finalement, nous avons montré quel type d'algorithme de renforcement semblait le plus prometteur pour ce cas d'application.

Pour résumer, cette thèse traite de l'apprentissage de bout-en-bout par renforcement fondé sur la fonction de valeur appliqué à la conduite autonome en milieu urbain.

Chapitre 3

Travail préliminaire : DRL sur un jeu de course réaliste

Ce projet a été effectué avant de commencer cette thèse. Il a ensuite abouti à une démonstration au CES (Consumer Electronics Show) 2017. Ce travail a en fait été le précurseur au sujet de thèse et c'est pourquoi il est pertinent de le rajouter au manuscrit en tant que travail préliminaire.

Les résultats de ce chapitre ont été présentés dans un article court au workshop CVPR 2017 [Perot et al., 2017] puis dans une version longue à la conférence ICRA 2018 [Jaritz et al., 2018].

Maximilian Jaritz, Raoul De Charette, Marin Toromanoff, Etienne Perot, and Fawzi Nashashibi. End-to-end race driving with deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2070–2075. IEEE, 2018.

3.1 Introduction

Les jeux de course ont été utilisés récemment [Mnih et al., 2016, Lillicrap et al., 2015] comme environnement pour de l'apprentissage par renforcement en tant que premier pas vers la conduite autonome. En effet, avant l'apparition des simulateurs comme CARLA [Dosovitskiy et al., 2017] ou Airsim [Shah et al., 2018] présentés dans le chapitre précédent, il n'y avait pas réellement d'environnement accessible aux chercheurs pour la conduite autonome. Utiliser un jeu de course déjà existant comme le jeu open-source TORCS [Wymann et al., 2000] était donc beaucoup plus simple que de créer un nouvel environnement à partir de rien. De plus, les jeux de course ont un objectif bien précis, atteindre l'arrivée le plus rapidement possible généralement sans considérer ni les collisions, ni de règles du code de la route. Ainsi le but poursuivi par l'agent est plus simple et il est bien plus facile d'évaluer l'agent que pour la conduite autonome. C'est pour ces raisons que les jeux de course semblent un très bon environnement "bac-à-sable" pour appliquer du renforcement à la voiture autonome.

(a) TORCS (b) WRC6

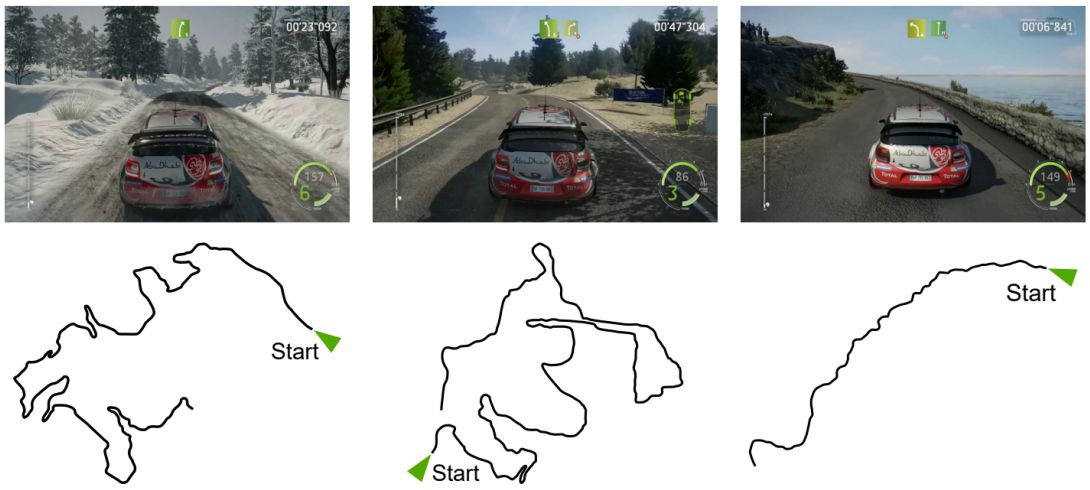
FIGURE 3.1 – Capture d'écran des jeux de course. (a) TORCS a des graphismes basiques et une physique simplifiée, et a été utilisé par des travaux connexes d'apprentissage par renforcement [Mnih et al., 2016, Lillicrap et al., 2015]. (b) WRC6 est un jeu de course moderne, sorti en 2016, qui essaye de rendre la physique du véhicule et le graphisme aussi réaliste que possible. Nous avons utilisé ce simulateur car il se rapproche plus d'un environnement réel et est beaucoup plus complexe à apprendre.

Dans ce chapitre, nous décrirons un travail effectué en amont de cette thèse consistant à appliquer un apprentissage de bout-en-bout sur un jeu de course réaliste. Nous utilisons en entrée de notre algorithme une unique image d'une caméra frontale posée sur le véhicule et calculons des commandes bas niveau (accélération, freinage, frein à main et angle au volant). Pour cela, nous utilisons l'algorithme A3C [Mnih et al., 2016] présenté dans le chapitre précédent et nous avons modifié l'espace d'actions et la fonction de récompense afin d'optimiser les performances pour le jeu de course. Nous avons utilisé le jeu vidéo World Rally Championship 6 (WRC6) [1]. Une comparaison de WRC6 avec le jeu open-source TORCS utilisé par la majorité des travaux précédents est montrée sur la Figure 3.1. WRC6 est bien plus réaliste que TORCS, WRC6 a en effet été développé

1. https://www.wrcthegame.com/

spécifiquement pour être aussi fidèle que possible en terme de physique (transfert de masse, adhérence de la route etc) et de graphisme. En utilisant un simulateur aussi réaliste, nous espérons faciliter le transfert de la simulation au monde réel.

Nous entraînons nos agents sur trois pistes différentes (c.f. Figure 3.2) pour une longueur totale de 29.6km. Les apparences visuelles (neige, montagne, bord de mer) et la physique (l'adhérence de la route) varient grandement entre ces différentes pistes. En utilisant un apprentissage distribué, nous entraînons sur les trois pistes en même temps pour apprendre une politique capable de conduire sous une grande variété de conditions.



(a) Snow (SE) 11.61km (b) Mountain (CN) 13.34km (c) Coast (UK) 4.59km

FIGURE 3.2 – Aperçus et contours des trois pistes de WRC6 utilisées pour l'entraînement de nos agents. Il y a des différences importantes que ce soit concernant l'aspect visuel ou le type de route : par exemple sur la neige, la voiture a tendance à glisser même à faible vitesse et le contrôle du véhicule est totalement différent par rapport aux deux autres pistes.

Nos contributions pour ce chapitre sont présentées ci-dessous :

— L'application d'un apprentissage par renforcement au jeu de course réaliste WRC6

— Introduction de différentes stratégies d'entraînement pour maximiser les bénéfices d'un apprentissage distribué (plusieurs pistes en parallèle, initialisation des agents variés).

— Une étude de différentes fonctions de récompenses permettant d'améliorer les performances de conduite.

En apprenant une seule politique, notre agent final est capable de conduire sur les trois pistes d'entraînement dans WRC6. De plus, nous avons observé une capacité de généralisation à quelques autres pistes jamais vues de WRC6. Cependant, comme nous le verrons en conclusion cette généralisation reste mitigée.

3.1.1 Travaux connexes : DRL pour les jeux de course

De nombreux travaux [Kondapalli et al., 2017, Aldape and Sowell, 2018, Mnih et al., 2016, Lillicrap et al., 2015] se sont intéressés à appliquer de l'apprentissage par renforcement à des jeux de course. Amazon a même créé le *AWS DeepRacer*[2], une interface de simulation entière pour entraîner des agents par renforcement sur une piste de course et participer à des compétitions entre les différents développeurs avec des voitures robotiques réelles (c.f. Figure 3.3). Malheureusement, le robot de Amazon n'était disponible qu'aux États-Unis et nous n'avons donc pas pu participer à cette compétition.

FIGURE 3.3 – La voiture robotique de AWS DeepRacer. L'idée est de fournir une interface pour d'abord entraîner les agents en simulation pour les faire ensuite concourir dans des compétitions réelles entre les différents participants.

Dans l'article de A3C [Mnih et al., 2016], les chercheurs ont évalué leur nouvel algorithme sur plusieurs environnements dont le jeu de course TORCS [Wymann et al., 2000]. Pour cela, Mnih et al. [2016] ont utilisé le même réseau de neurones et les mêmes hyperparamètres que pour les jeux Atari et ont discrétisé les actions d'accélération et d'angle au volant. L'idée étant de démontrer l'intérêt et le gain en performance sur des tâches aussi diverses que possible entre leur nouvel algorithme A3C et DQN [Mnih et al., 2015] la référence en DRL à l'époque. Ils ont ainsi pu montrer que sur toutes les tâches évaluées, i.e. les jeux Atari et le jeu de course TORCS, A3C avait des meilleures performances que DQN et devenait ainsi le nouvel état de l'art en apprentissage par renforcement. C'est pour cette raison que nous avons utilisé cet algorithme pour notre travail. Tout au long de ce chapitre, nous utiliserons leur approche sur TORCS (architecture, fonction de récompense) comme référence après l'avoir ré-entraîné sur le jeu de course réaliste WRC6 et nous la comparerons avec notre propre fonction de récompense.

3.2 Méthode

Dans ce chapitre, notre but est d'effectuer un apprentissage de bout-en-bout sur un jeu de rallye où la physique de la route et les apparences visuelles changent pour chaque piste, par exemple une route enneigée glissante dans une forêt en hiver ou une route avec des virages serrés sous un temps ensoleillé en montagne. Cette tâche est complexe

2. https://aws.amazon.com/deepracer/

car il faut non seulement apprendre à conduire mais aussi implicitement apprendre la dynamique du véhicule qui dépend des conditions extérieures de chaque piste.

L'agent doit apprendre le contrôle total du véhicule - angle au volant, frein, accélération et même frein à main pour les dérapages - pour conduire dans le jeu WRC6. Nous utilisons une API (pour *Application Programming Interface*) pour communiquer les images, les métadonnées (la vitesse courante ou la rotation du véhicule par exemple) et les actions prises par l'agent entre l'algorithme de RL et le simulateur. Cette API de communication avec WRC6 a été spécifiquement développée pour notre besoin par Kylotonn, l'entreprise développant le jeu. L'architecture générale de notre apprentissage de bout-en-bout est illustrée en Figure 3.4.

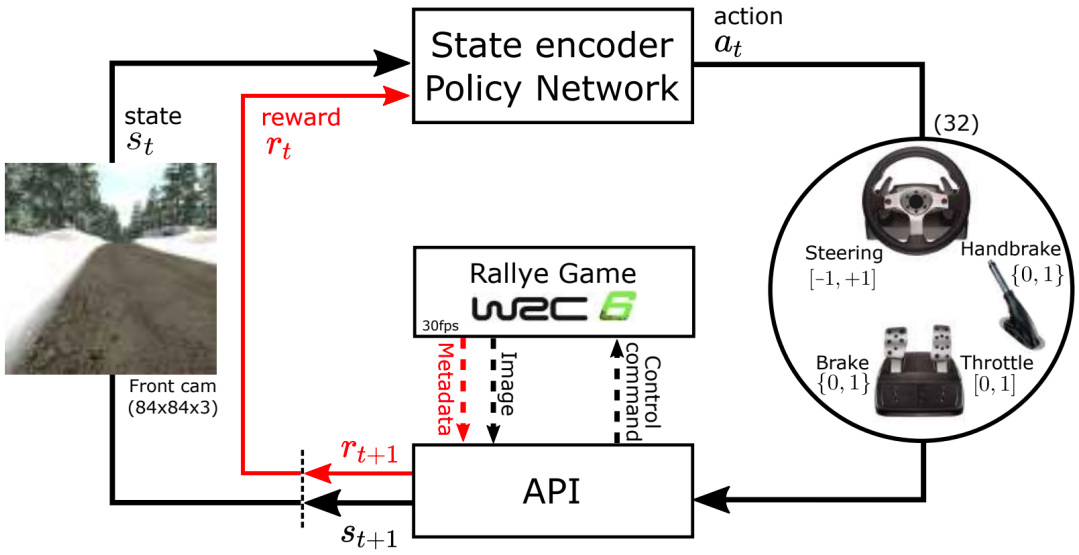


FIGURE 3.4 – Représentation de notre architecture d'apprentissage de bout-en-bout utilisant une API pour les communications entre le jeu WRC6 et notre algorithme de renforcement (les flèches rouges ne sont utilisées que durant l'entraînement). Notre réseau de neurones cherche à apprendre un contrôle optimal en utilisant seulement une image 84x84 d'une caméra frontale sur le véhicule et la vitesse courante. L'environnement est complexe avec une physique et des graphismes réalistes.

A chaque pas de temps, l'agent reçoit l'état du jeu s_t sous la forme d'une image et de la vitesse courante, agit sur le véhicule avec une action a_t et finalement obtient une récompense r_{t+1}. Comme indiqué précédemment, nous utilisons l'algorithme A3C [Mnih et al., 2016] pour optimiser une politique qui calcule la probabilité des actions pour le contrôle du véhicule en utilisant seulement l'image RGB d'une caméra frontale.

Dans cette section, nous décrirons dans un premier temps comment nous avons distribué notre apprentissage sur plusieurs machines. Puis nous détaillerons notre approche concernant la représentation des actions et de la fonction de récompense.

3.2.1 Apprentissage distribué sur plusieurs machines

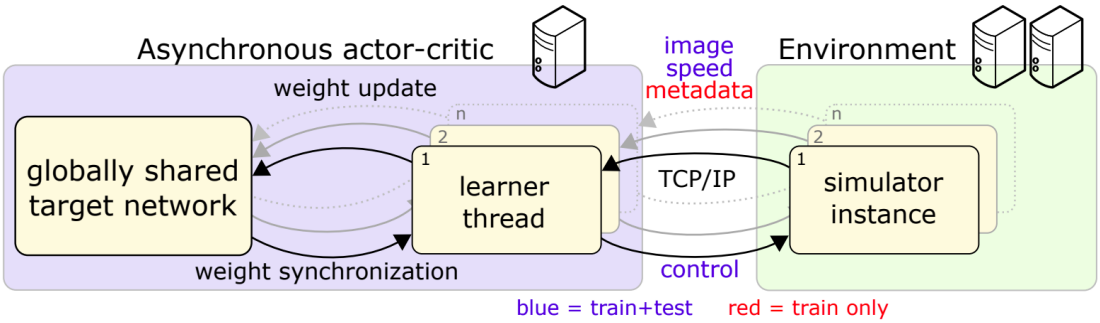


FIGURE 3.5 – Schéma de notre architecture d'apprentissage distribuée. Plusieurs instances de WRC6 tournent sur deux machines (encadré vert) et communiquent via une API dédiée avec les threads des agents qui tournent sur une machine différente (encadré bleu). Les agents mettent à jour de manière asynchrone les poids du réseau de neurones cible partagé. Les agents vont aussi synchroniser régulièrement leur propre réseau de neurones avec le réseau de neurone partagé.

Comme mentionné dans le chapitre précédent, l'algorithme A3C utilise plusieurs agents en parallèle ce qui permet de décorréler les états successifs tout en gardant le caractère *on-policy* de l'architecture acteur-critique. Mnih et al. [2016] ont aussi introduit une méthode permettant de partager les gradients calculés par chacun des agents afin d'optimiser le même réseau de neurones partagé, appelé réseau cible. Dans notre implémentation, les différents agents sont exécutés sur des threads séparés pour faire de l'apprentissage en parallèle. Notre configuration d'entraînement est schématisée en Figure 3.5. L'algorithme A3C (encadré bleu) est lancé sur un PC central où les différents agents peuvent mettre à jour de manière asynchrone les poids du réseau de neurones partagé. L'environnement (encadré vert) s'exécute sur d'autres machines (deux autres PC dans notre cas). Chaque agent communique avec sa propre instance du simulateur avec une API qui utilise le réseau LAN via un protocole TCP/IP. Dans notre cas, nous avons entraîné neuf agents (associés à neuf instances de WRC6) repartis sur les trois pistes d'entraînement. Cette configuration multi-machines est nécessaire car WRC6 exige bien plus de puissance de calcul que TORCS ou les jeux Atari et qu'il n'est pas possible de lancer un grand nombre d'instances de WRC6 sur un seul PC. De plus, cette approche de parallélisation permet d'entraîner sur différentes pistes en même temps pour améliorer la capacité de généralisation de la politique finale.

3.2.2 Espace d'actions

Dans WRC6, l'agent doit apprendre les commandes pour contrôler l'angle au volant dans l'intervalle [-1, 1], l'accélération dans l'intervalle $[0, 1]$, le frein avec une valeur binaire dans $\{0, 1\}$ et le frein à main avec une valeur binaire dans $\{0, 1\}$. Le frein à main a été rajouté pour permettre à l'agent d'apprendre à déraper pour prendre les virages plus

	Commandes de contrôle			
Nb d'actions	angle au volant	accélération	frein	frein à main
27	$\{-1., -0.75, ..., 1.\}$	$\{0.0, 0.5, 1.0\}$	$\{0\}$	$\{0\}$
4	$\{-1., -0.5, 0.5, 1.\}$	$\{0.0\}$	$\{0\}$	$\{1\}$
1	$\{0.0\}$	$\{0.0\}$	$\{1\}$	$\{0\}$

TABLE 3.1 – Les 32 actions disponibles pour notre agent appris par renforcement. La première colonne indique le nombre d'actions possibles avec les différents ensembles de valeurs possibles pour les quatre commandes. Nous avons sur-représenté le nombre d'actions sans frein pour que l'agent ne reste pas bloqué à sa position initiale lorsqu'il agit aléatoirement (particulièrement au début de l'apprentissage).

rapidement que si seul le frein standard était disponible. Nous avons choisi de discrétiser l'ensemble de ces commandes en un espace de 32 actions listées dans la Table 3.1. Le nombre d'actions a été choisi arbitrairement mais en faisant bien attention à attribuer des angles au volant variés aux actions avec frein à main pour permettre à l'agent de déraper. Nous avons aussi choisi de prendre bien plus d'actions sans frein qu'avec pour éviter le problème de l'agent bloqué à sa position initiale lorsqu'il agit aléatoirement (au début de l'apprentissage). En effet, la friction statique est modélisée de manière très réaliste dans WRC6 et il faut appliquer une accélération pendant un certain temps avant que le véhicule ne commence à avancer. L'idéal aurait été de démarrer l'agent à des vitesses aléatoires mais cela n'était pas permis par le jeu pour des raisons techniques. Nous verrons en conclusion, que nous avons dû faire face à de nombreuses limitations techniques avec le jeu WRC6 tout au long de ce travail.

3.2.3 Fonction de récompense

Le temps nécessaire pour atteindre l'arrivée de la piste est le seul score dans un jeu de course. Malheureusement, ce signal est très peu fréquent et il est très difficile d'apprendre par renforcement avec une fonction de récompense aussi clairsemée. C'est pour cette raison que Mnih et al. [2016] et Lillicrap et al. [2015] ont utilisé une fonction de récompense beaucoup plus dense. A chaque pas de temps, la récompense est proportionnelle à la vitesse courante v_t de l'agent multipliée par le cosinus de l'angle α_t de l'agent par rapport à sa voie, cf équation 3.1.

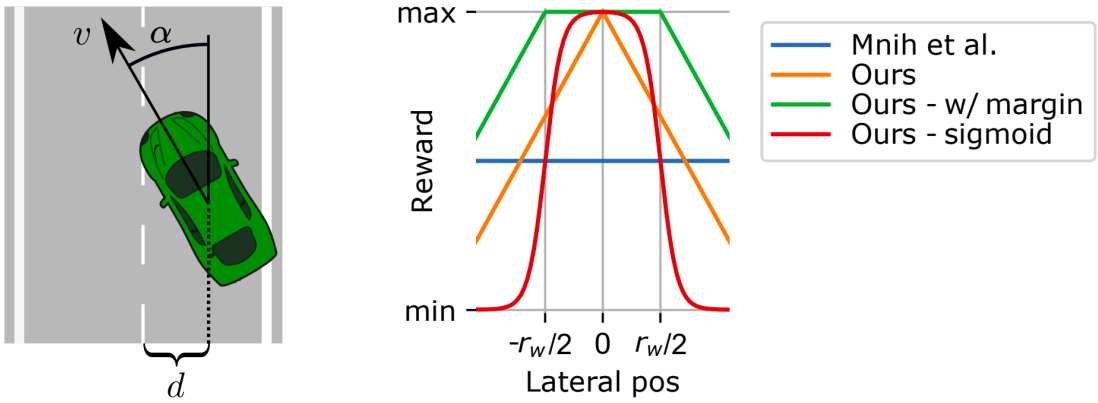
$$r_t = v_t \cos \alpha_t \tag{3.1}$$

Dans nos premiers tests, nous avons découvert que cette fonction de récompense n'incitait pas l'agent à éviter de glisser contre les rambardes sur le bord des voies. L'agent apprenait rapidement à se laisser guider par les rambardes sans avoir réellement besoin de diriger le véhicule. Pour résoudre ce problème, nous avons ajouté la distance au centre de la voie d, illustrée en Figure 3.6, comme pénalité au calcul de notre récompense, cf équation 3.2. Pour limiter la récompense à l'intervalle $[-1, 1]$, nous normalisons la vitesse avec la vitesse maximale observée empiriquement dans le jeu, i.e nous divisons la vitesse

v_t par v_{max} = 150km/h. Nous normalisons aussi la distance d_t en la divisant par la largeur courante de la route.

$$r_t = v_t(\cos \alpha_t - |d_t|). \tag{3.2}$$

Notre récompense forme donc un triangle qui est maximum lorsque l'agent est parfaitement au centre de sa voie. Nous avons aussi entraîné avec deux autres formes proches de cette récompense, la première avec un plateau constant tant que l'agent est proche du centre de la voie (cf Figure 3.6, *w/ margin*) et la seconde en utilisant une fonction sigmoïde pour retirer la discontinuité en 0 (cf Figure 3.6, *sigmoid*). Ces récompenses permettent d'inciter l'agent à rester au centre de sa voie et à réellement contrôler l'angle au volant sans se faire guider par les rambardes.



(a) Metadata (b) Reward functions

FIGURE 3.6 – Représentation des différentes fonctions de récompense. (a) Pour calculer la récompense, nous utilisons certaines métadonnées : la vitesse courante v, l'angle α entre l'agent et sa voie ainsi que la distance d au centre de la voie. (b) Différentes fonctions de récompenses en fonction de la distance latéral entre l'agent et le centre de la voie. r_w correspond à la largeur de la route actuelle.

3.3 Expériences

Dans cette section, nous allons brièvement détailler notre configuration d'entraînement, les métriques que l'on a utilisées pour évaluer nos agents ainsi que des résultats quantitatifs et qualitatifs de notre agent final.

Configuration d'entraînement Pour accélérer la génération de données, nous avons retiré certains effets graphiques coûteux et utilisé un champ de vision plus faible comparé à la vue standard du jeu. Le temps de simulation est de 30 images par seconde et le moteur de jeu attend de recevoir la prochaine action avant de passer au pas de temps suivant. En pratique, le jeu tourne plus vite que le temps réel et atteint environ 100

images par seconde durant l'entraînement, i.e. 3 fois plus rapide que le temps réel. Cela signifie qu'en utilisant 9 instances du jeu, on peut simuler environ 30 jours de jeu en 1 jour de temps réel.

Métriques Nous avons utilisé trois métriques indépendantes de la fonction de récompense pour évaluer nos agents. Les métriques sont calculées par épisode. Dans notre cas, l'épisode commence lorsque l'agent est positionné sur la piste et s'arrête lorsque l'agent atteint la fin de la course ou se retrouve bloqué contre les rambardes.

La première métrique est la distance parcourue par l'agent durant l'épisode. Les deux autres métriques sont la vitesse moyenne de l'agent durant l'épisode ainsi que le nombre de collisions par kilomètre. La distance parcourue donne une mesure de la performance globale de l'agent durant l'entraînement. Les deux autres métriques sont utilisées pour différencier différents styles de conduite.

Résultats quantitatifs et qualitatifs de notre agent entraîné Les résultats quantitatifs sur chacune des trois pistes d'entraînement et pour chaque métrique sont représentés sur la Figure 3.7. Une vidéo de notre agent en train de conduire dans WRC6 pour avoir des résultats qualitatifs du comportement de notre agent peut se trouver ici : https://www.youtube.com/watch?v=AF0sryuSHdY&feature=youtu.be&t=118.

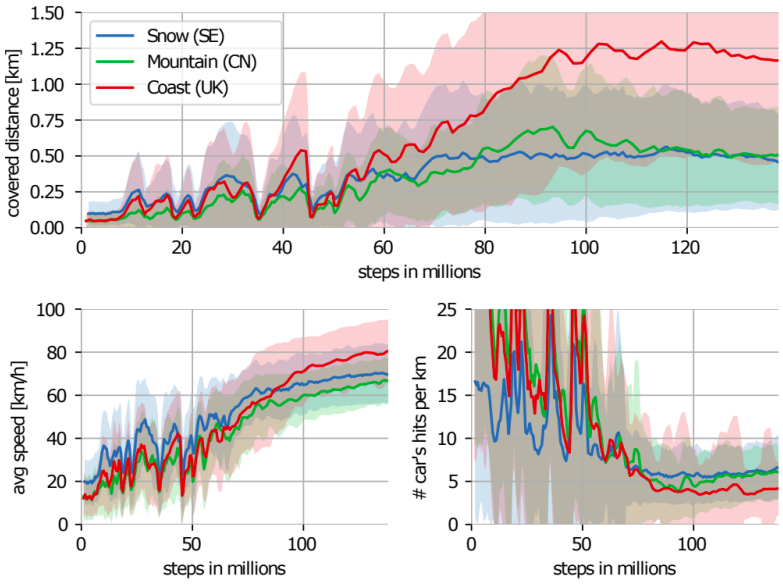


FIGURE 3.7 – Les performances de notre agent en fonction du nombre de pas d'entraînement pour chaque métrique et chacune des trois pistes d'entraînements. La moyenne est représentée en trait plein tandis que la déviation standard en clair. L'agent a beaucoup plus de difficulté sur les pistes *snow* et *mountain* car elles contiennent bien plus de virages serrés et que la route enneigée a une physique glissante difficile à contrôler.

Impact du choix de la fonction de récompense Dans ce paragraphe, nous comparons la fonction de récompense de Mnih et al. [2016] (équation 3.1) avec nos différentes fonctions de récompense (équation 3.2). Les résultats de cette expérience peuvent se trouver sur la Table 3.2. On constate bien l'intérêt de notre fonction de récompense sur le nombre de collisions, l'agent entraîné avec notre récompense effectue quasiment quatre fois moins de collisions que l'agent entraîné avec la fonction de récompense de Mnih et al. [2016]. Par contre, on constate que notre fonction de récompense réduit la vitesse moyenne de l'agent, ce qui semble logique car l'agent prend plus de précautions dans les virages pour ne pas trop s'éloigner du centre de sa voie. Concernant les deux autres formes de notre fonction de récompense, *w/ margin* et *sigmoid*, on constate que les résultats sont très similaires à ceux obtenus avec notre fonction de récompense en triangle. Il est important de noter que chaque expérience nécessitait plus d'une semaine sur nos trois machines de calcul et cela a grandement limité le nombre d'expériences que l'on a pu effectuer pour essayer d'améliorer ce compromis entre vitesse et nombre de collisions.

Fonction de récompense	vitesse moyenne [km/h]	collisions [hits/km]
Mnih et al. (Eq. 3.1)	**106.9**	9.26
Notre fonction (Eq. 3.2)	91.4	2.25
Notre fonction *w/ margin*	100.3	5.95
Notre fonction *sigmoid*	89.8	**2.24**

TABLE 3.2 – Moyenne sur tous les épisodes d'évaluation (à 130M de pas d'entraînement) de la vitesse et du nombre de collisions. Notre fonction de récompense résulte en un agent effectuant beaucoup moins de collisions mais qui du coup va légèrement moins vite en moyenne.

Étude de la généralisation Nous avons vu que notre agent était capable de conduire sur les trois pistes d'entraînement ce qui démontre une certaine capacité de généralisation car les épisodes ne sont jamais strictement identiques WRC6 étant un environnement stochastique (les animations des objets, les changements d'illumination etc). Cependant, nous avons évidemment voulu tester notre agent sur d'autres pistes de WRC6 que l'agent n'avait réellement jamais vu. Les résultats ont été très mitigés, sur la grande majorité des autres pistes de WRC6, l'agent échouait totalement à conduire... Cependant, l'agent arrivait à conduire raisonnablement sur les autres pistes avec des environnements similaires, particulièrement les pistes en montagne car elles ont toutes le même style de marquage au sol jaune. En conclusion, on peut dire que les capacités de généralisation de notre approche était plutôt faibles et qu'on est très loin d'un possible transfert sur données réelles.

3.4 Discussion : Qu'a apporté ce travail préliminaire à la thèse ?

Dans cette section, nous allons tout d'abord énoncer et discuter de certains choix de ce travail préliminaire qui ne semblent pas forcément les plus pertinents après les trois ans de cette thèse. Dans un deuxième temps, nous indiquerons quels ont été les éléments importants et les enseignements que l'on a retirés de ce premier travail en apprentissage par renforcement.

Tout d'abord, il est important de noter que les articles appliquant de l'apprentissage par renforcement à la voiture autonome sont assez rares aujourd'hui et ils l'étaient d'autant plus lorsque ce travail a été effectué (2016 et 2017). Les simulateurs introduits dans le chapitre précédent comme CARLA [Dosovitskiy et al., 2017] ou Airsim [Shah et al., 2018] n'existaient pas encore. C'est pour cela que ce travail était à l'époque assez novateur dans son application. En particulier, il a été utilisé par Valeo comme démonstration au CES 2017 ainsi que dans de nombreux autres salons de technologie comme VivaTech 2018 à Paris. Cela montre bien l'intérêt de ce travail qui a certainement contribué au lancement par la suite de cette thèse par Valeo. Cependant, ces travaux ont été menés avant d'avoir fait l'état de l'art général sur l'apprentissage par renforcement et c'est pourquoi ils comportent des faiblesses surtout à la lumière des trois années de thèse effectuées après.

3.4.1 Description des faiblesses de ce travail préliminaire

Choix de l'algorithme A3C et de l'espace d'actions Tout d'abord le choix de l'algorithme A3C [Mnih et al., 2016], algorithme acteur-critique *on-policy*, n'était pas nécessairement le choix le plus approprié particulièrement en sachant que le plus grand obstacle à ces travaux était le temps d'entraînement des agents : presque une semaine sur trois machines même en réduisant considérablement le rendu graphique du jeu WRC6. Il fallait en effet environ 100 millions d'images d'entraînement pour atteindre des bonnes performances, cela aurait pu certainement être réduit en utilisant un algorithme *off-policy* comme DQN [Mnih et al., 2015] ou ses améliorations [Van Hasselt et al., 2016, Schaul et al., 2015, Hessel et al., 2018] pouvant ré-utiliser plusieurs fois chaque expérience grâce à une mémoire de reprise. De plus, ce temps extrêmement long pour effectuer les entraînements a grandement réduit le nombre d'expériences possibles et ainsi limité les contributions de ce travail.

Finalement, l'un des avantages des algorithmes acteur-critique comparés aux algorithme fondés sur la fonction de valeur est qu'ils peuvent s'appliquer avec un espace d'actions continu. Il semble donc qu'utiliser un algorithme acteur-critique avec des actions discrètes ne soit pas un choix pertinent.

Choix du simulateur WRC6 Le jeu WRC6 a certainement des meilleurs graphismes et une physique plus réaliste que TORCS voire même que CARLA. Cependant, il y a de nombreux inconvénients à l'utiliser. Comme décrit dans le chapitre précédent, le critère

principal que doit vérifier un bon simulateur pour l'apprentissage par renforcement est la flexibilité de l'environnement. En fait, WRC6 a les mêmes désavantages que GTA, le jeu est extrêmement lourd à faire tourner, la génération de données est très coûteuse et surtout chaque nouveau développement est très complexe car le jeu n'a été pas été développé dans le but de faire de l'apprentissage. Ainsi, nous devions rentrer en discussion avec Kylotonn (le développeur du jeu) à chaque changement sur le moteur de jeu. Nous n'avons ainsi jamais pu démarrer nos agents à des endroits aléatoires sur les pistes et nous n'avons pas pu changer le nombre d'images par seconde ou le temps de simulation. Il était aussi impossible d'entraîner plusieurs agents en parallèle sur la même instance du jeu ce qui aurait pu faciliter la génération de données. De plus, comme mentionné précédemment, nous avons dû retirer la grande majorité des aspects graphiques et nous n'utilisons qu'une image 84x84 ce qui limite grandement l'intérêt d'avoir un simulateur avec des graphismes réalistes. En conclusion, comme indiqué dans le chapitre précédent pour GTA, le jeu WRC6 ne semble pas un simulateur approprié pour faire de l'apprentissage par renforcement et il faudrait lui préférer des simulateurs plus légers et flexibles comme CARLA, Airsim voire TORCS...

3.4.2 Les enseignements tirés de ce travail préliminaire

Malgré les faiblesses de ce travail initial, de nombreux enseignements intéressants peuvent en être tirés. Tout d'abord, cela m'a permis de réaliser la difficulté à appliquer de l'apprentissage par renforcement sur une tâche concrète. En effet, cela a nécessité énormément de temps rien que pour obtenir un agent légèrement capable de conduire et de suivre grossièrement sa voie. De plus, cela nous a fait prendre conscience de la quantité de données et du temps nécessaire pour apprendre même un comportement relativement simple et la nécessité d'avoir une génération de données efficace et rapide. Finalement, nous avons pu voir que la généralisation en apprentissage par renforcement était encore plus complexe que pour l'apprentissage supervisé et dans notre cas quasiment inexistante. Cela a nécessité tellement de temps et d'effort pour avoir un agent capable de conduire sur les pistes d'entraînement, que cela nous a permis de nous rendre compte du gouffre à franchir avant de réussir à entraîner un agent capable de véritable généralisation (sur une piste réellement "différente") sans même parler du transfert de la simulation au monde réel. En conclusion, ce travail a été réellement passionnant, il m'a permis de découvrir les difficultés inhérentes à l'apprentissage par renforcement et il a été la raison principale de ma motivation à faire cette thèse.

3.5 Conclusion

Dans ce chapitre, nous avons introduit un travail préliminaire à cette thèse consistant à appliquer un apprentissage de bout-en-bout par renforcement à un jeu de course avec des graphismes et une physique réaliste. Nous avons réussi à obtenir un agent capable de conduire sur trois pistes avec des apparences visuelles et des physiques très variées. Même si les contributions réelles de ce travail sont restreintes, il m'a permis de prendre

connaissance des nombreuses difficultés pour appliquer de l'apprentissage par renforcement : le temps extrêmement long d'apprentissage, l'immense quantité de données nécessaire pour obtenir une politique raisonnable même simple ou la difficulté d'obtenir un agent capable de généraliser à d'autres domaines que ceux vu durant l'apprentissage. Surtout, il est important de noter que ce travail a été effectué avant de faire un véritable état de l'art de l'apprentissage par renforcement d'où des choix parfois peu pertinents, que ce soit sur le choix de l'algorithme ou bien sur l'environnement de simulation. Dans la suite de ce manuscrit, nous introduirons les travaux réellement effectués durant cette thèse qui ont grandement bénéficié de cette première expérience et de ses potentielles erreurs. En effet, nous essayerons de répondre à la plupart des questions soulevées par ce premier travail préliminaire : création d'un nouvel algorithme, Rainbow-IQN Ape-X (c.f. Chapitre 4), fondé sur la fonction de valeur pour être plus efficace en données, utilisation d'*indices implicites* (c.f. Chapitre 5) pour aller au delà d'une image 84x84 et gérer des tâches plus complexes et finalement introduction d'un simulateur fondé sur des images réelles (c.f. Chapitre 6 et 7) pour l'application sur un véhicule et des données réelles.

Chapitre 4

Nouvel algorithme état de l'art sur Atari : Rainbow-IQN Ape-X

Les contributions de ce chapitre ont été publiées dans [Toromanoff et al., 2019] :

Marin Toromanoff, Emilie Wirbel, and Fabien Moutarde. Is deep reinforcement learning really superhuman on atari? In *Deep Reinforcement Learning Workshop of 39th Conference on Neural Information Processing Systems (Neurips' 2019)*, 2019.

Le code de l'algorithme développé est accessible en open-source :

https://github.com/valeoai/rainbow-iqn-apex

4.1 Introduction

Dans le chapitre précédent, nous avons introduit un travail préliminaire à cette thèse consistant à appliquer l'algorithme A3C [Mnih et al., 2016] à un jeu de course avec des graphismes et une physique réaliste. Nous avons aussi ajouté en discussion finale que le choix d'un algorithme fondé sur la politique ne semblait pas le plus adapté pour l'application à la conduite autonome en particulier de par son caractère *on-policy* et sa moindre *efficacité en données* . Dans ce chapitre, nous allons donc présenter l'algorithme que nous avons développé en prenant en compte les remarques et les suggestions du Chapitre 2, c'est à dire utiliser un algorithme *off-policy* fondé sur la fonction de valeur. Cet algorithme est en fait la combinaison de trois articles majeurs en apprentissage par renforcement fondé sur la fonction de valeur, l'algorithme Rainbow [Hessel et al., 2018] qui avait les meilleures performances sur le benchmark Atari [Bellemare et al., 2013] au moment de ce travail, l'algorithme IQN (pour *Implicit Quantiles Network*) [Dabney et al., 2018] ainsi que l'article Ape-X [Horgan et al., 2018] qui introduit une architecture générale pour distribuer et rendre parallélisable les algorithmes de renforcement fondés sur la fonction de valeur qui étaient jusqu'alors majoritairement à *agent unique*.

L'environnement ALE [Bellemare et al., 2013] (pour *Arcade Learning Environment*) regroupe plus de 60 jeux Atari différents et est l'un des benchmarks les plus utilisés pour évaluer et comparer différents algorithmes de renforcement. Pour développer notre nouvel algorithme, s'assurer de ses bonnes performances et pouvoir le comparer à l'état de l'art, nous avons nous aussi utilisé ce benchmark. C'est pourquoi dans ce chapitre nous nous intéresserons exclusivement au contrôle d'un agent dans les jeux Atari. L'environnement ALE [Bellemare et al., 2013] contient de nombreuses tâches différentes allant du simple contrôle 2D d'une plateforme dans le jeu Pong à l'exploration d'un labyrinthe très complexe comme dans Montezuma's Revenge qui reste aujourd'hui un jeu où les algorithmes *généraux* de DRL ont des performances extrêmement faibles. Lorsque nous parlerons d'algorithmes *généraux*, cela signifie des algorithmes utilisant le moins possible voire aucune information spécifique à l'environnement. L'idée de ce type d'algorithme est de pouvoir être appliqué sur n'importe quelle tâche sans travail supplémentaire, ainsi dans ce chapitre nous nous intéresserons exclusivement aux algorithmes ayant exactement la même architecture et les mêmes hyperparamètres pour l'ensemble des jeux Atari.

Suite aux succès récents du DRL et en particulier de DQN [Mnih et al., 2015], il y a eu de plus en plus de contributions et d'innovations sur le benchmark Atari. Cela a pour effet qu'il est de plus en plus difficile de faire des comparaisons équitables entre différents algorithmes. En particulier, il y a des différences significatives dans les procédures d'entraînement et d'évaluation des algorithmes entre différentes publications. Ce problème est exacerbé par le fait qu'entraîner des agents par DRL est extrêmement lourd en temps et en ressources de calcul, ce qui met un grand obstacle à la ré-évaluation des travaux précédents. Plus précisément, même si l'émulateur Atari peut s'exécuter extrêmement rapidement, entraîner un agent sur un jeu Atari nécessite environ une semaine sur un bon GPU (comme une GTX 1080TI) ce qui implique plus d'un an de calcul sur un seul GPU pour entraîner un agent sur chacun des 61 jeux Atari. Une standardisation des

procédures d'évaluation est nécessaire pour faire du *DRL that matters* comme indiqué par Henderson et al. [2018] pour le benchmark MuJoCO [Todorov et al., 2012] : les auteurs critiquent le manque de reproductibilité et discutent des points à améliorer pour permettre d'effectuer des comparaisons équitables et consistantes entre les différents articles.

Dans ce chapitre, nous présenterons d'abord différents problèmes dans les procédures d'évaluation de différents algorithmes de DRL sur ALE et quels sont leurs impacts. Nous proposerons ensuite un nouveau standard d'évaluation, qui étend les recommandations de Machado et al. [2018], que nous avons appelé SABER (pour *Standardized Atari BEnchmark for Reinforcement learning*). Nous suggérons aussi de comparer les performances des agents à la référence des records du monde humains et nous montrerons ainsi que les algorithmes de renforcement sont en fait très loin de résoudre la majorité des jeux Atari. Afin de donner un exemple concret de notre nouveau standard d'évaluation SABER, nous avons ré-évalué l'algorithme Rainbow [Hessel et al., 2018] qui atteignait les meilleures performances sur ce benchmark au moment de ce travail. Finalement, nous introduirons et évaluerons avec SABER un algorithme atteignant un nouvel état de l'art : une combinaison distribuée et parallélisable de Rainbow et de IQN [Dabney et al., 2018].

Nous résumons nos contributions principales de ce chapitre ci-dessous :

— L'introduction, la description et la justification d'un nouveau standard d'évaluation sur le benchmark Atari : SABER (pour *Standardized Atari BEnchmark for Reinforcement learning*).

— Introduction d'une nouvelle référence, la référence des records du monde humains. Nous argumentons que cette référence est bien plus représentative du niveau humain que celle utilisée dans la plupart des travaux précédents. Avec cette référence, nous montrerons que le benchmark Atari est en fait une tâche extrêmement complexe pour les algorithmes généraux actuels.

— Une évaluation de l'algorithme Rainbow, le plus performant sur Atari au moment de notre publication, avec notre nouveau standard d'évaluation SABER.

— Un nouvel algorithme Rainbow-IQN dépassant les performances de Rainbow ainsi qu'une comparaison sur SABER afin de donner un ordre de grandeur d'amélioration pour les futurs travaux et les futures comparaisons.

— Pour des raisons de reproductibilité, une implémentation open-source de Rainbow et de Rainbow-IQN distribuée selon le principe introduit dans l'article de Horgan et al. [2018].

4.2 Travaux connexes

Nos contributions s'intéressent d'abord à comment rendre les travaux de DRL plus reproductibles et effectuer des comparaisons les plus équitables possibles entre les différentes publications. C'est pourquoi nous allons d'abord introduire certains articles qui alertent sur le fait que la majeure partie des travaux de DRL ne sont pas reproductibles et ne permettent pas d'effectuer des comparaisons équitables. Dans une deuxième partie,

nous décrirons les différents algorithmes fondés sur la fonction de valeur qui ont permis de construire notre propre algorithme de renforcement : Rainbow IQN Ape-X.

4.2.1 Reproductibilité et comparaison dans le DRL

4.2.1.1 *Deep Reinforcement Learning that matters* [Henderson et al., 2018]

L'article *Deep Reinforcement Learning that matters* [Henderson et al., 2018] est l'un des premiers travaux à alerter sur une crise de reproductibilité dans le domaine de l'apprentissage par renforcement profond. Cet article s'appuie sur le benchmark MuJoCo [Todorov et al., 2012], un benchmark comportant différentes tâches de contrôle robotique, lui aussi extrêmement utilisé pour comparer les algorithmes de renforcement. Henderson et al. [2018] montrent que les algorithmes de DRL sont très instables et extrêmement sensibles aux hyperparamètres, à l'architecture choisie et même à la base de code open-source utilisée pour le même algorithme ! En effet, en évaluant l'algorithme PPO [Schulman et al., 2017] sur la même tâche mais en prenant différentes implémentions open-source, ils ont obtenu des résultats totalement différents. De plus, les chercheurs critiquent des pratiques couramment utilisées dans la littérature qui peuvent biaiser les résultats rapportés. L'exemple le plus frappant est la méthode régulièrement utilisée consistant à rapporter le résultat des N meilleurs essais, ce qui biaisent évidemment les résultats, surtout que parfois il n'est même pas indiqué les N meilleurs parmi combien d'expériences... Un autre exemple extrêmement intéressant montré par Henderson et al. [2018] est qu'on peut artificiellement donner l'impression d'avoir de meilleurs résultats juste en modifiant la graine du générateur aléatoire ! Ce problème est illustré en Figure 4.1.

Tous ces problèmes soulevés par Henderson et al. [2018] montrent à quel point il est difficile de bien comparer différents algorithmes de DRL et qu'il est nécessaire de standardiser les méthodes d'évaluation et de retirer les méthodes pouvant ajouter du biais comme rapporter les résultats des meilleurs essais. Les chercheurs insistent surtout sur le fait qu'il faut absolument que tous les hyperparamètres, les détails d'implémentation et la configuration expérimentale soit rapportés avec chaque article : l'idéal étant de donner le code en open-source contenant tous les processus d'entraînement et d'évaluation.

4.2.1.2 *Revisiting ALE* [Machado et al., 2018]

L'article de Machado et al. [2018] quant à lui s'intéresse spécifiquement à la reproductibilité et l'évaluation sur le benchmark Atari. Machado et al. [2018] décrivent des divergences dans les procédures d'entraînement et d'évaluation et expliquent comment cela peut amener des difficultés pour comparer différents algorithmes. Ils vont ainsi proposer un premier ensemble de recommandations pour standardiser ces procédures, ces recommandations constituant d'ailleurs le fondement de notre nouveau standard SABER. Ces recommandations concernent plusieurs points soit durant l'entraînement des agents, soit pour leur évaluation. Nous allons les présenter dans les paragraphes suivants.

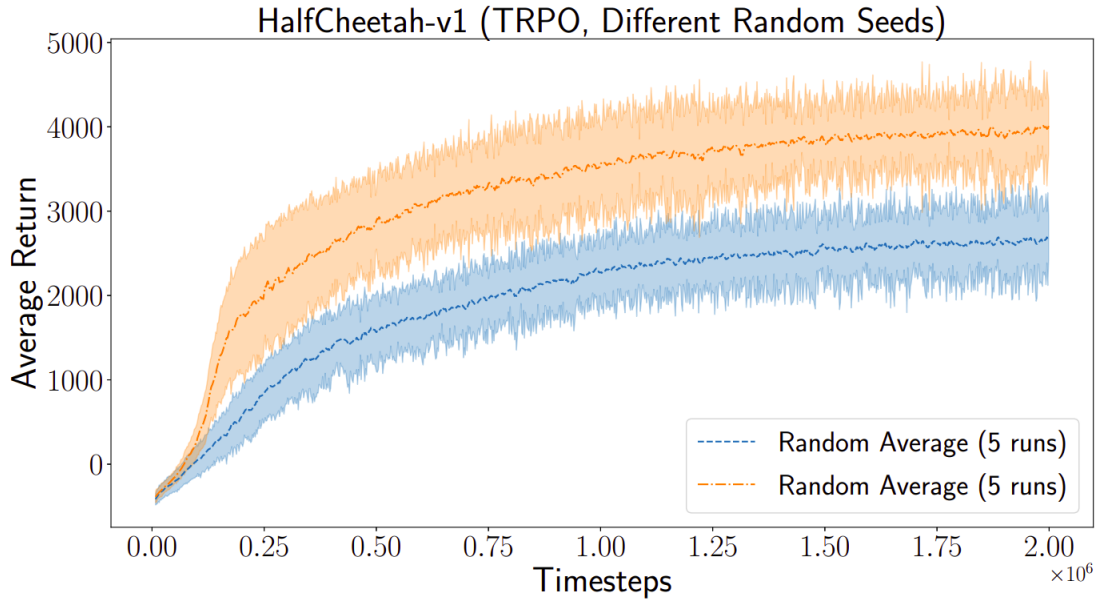


FIGURE 4.1 – Évaluation du même algorithme (PPO) avec exactement les mêmes hyperparamètres et base de code sur la tâche HalfCheetah du benchmark MuJoCo. Les deux évaluations sont moyennées à partir de 5 graines (*random seed*) différentes. On constate donc qu'on a l'impression d'avoir deux évaluations statistiquement différentes alors que le seul et unique paramètre qui change et la graine génératrice ! Image provenant de Henderson et al. [2018]

Stochasticité L'environnement ALE est totalement déterministe, c'est à dire que les mêmes actions à partir de l'état initial arriveront toujours exactement aux mêmes états avec les mêmes récompenses. Cela est en fait extrêmement problématique pour l'évaluation d'algorithmes généraux. Par exemple, un algorithme apprenant par coeur une bonne trajectoire peut atteindre des scores très importants même dans un comportement en boucle ouverte. Cela est donc bien loin de l'idée initiale qui est d'apprendre un algorithme général pouvant gérer plusieurs situations en même temps et qui sache réagir à ses propres erreurs ou à des variations de l'environnement. Pour pallier ce problème, Machado et al. [2018] ont introduit les *sticky actions* : les actions de l'agent sont répétées avec une probabilité ξ ce qui amène un comportement non déterministe. Ils montrent ainsi que les sticky actions réduisent énormément les performances d'un algorithme qui exploite le déterminisme de l'environnement mais n'ont pas d'impact significatif sur des algorithmes apprenant des politiques plus robustes comme DQN Mnih et al. [2015]. Nous avons utilisé des sticky actions avec une probabilité $\xi = 0.25$ dans toutes nos expériences.

Fin d'épisode : Utiliser le signal de game over Dans la plupart des jeux Atari, le joueur possède plusieurs vies et le jeu est véritablement terminé que quand toutes les vies sont perdues. Mais dans certains articles [Mnih et al., 2015, Hessel et al., 2018, Dabney

et al., 2018], l'épisode termine après la perte de la première vie durant l'entraînement et après la perte de toutes les vies durant l'évaluation. Cela permet en fait d'aider l'agent à apprendre comment éviter de perdre des vies et amène des comparaisons inéquitables avec les agents qui n'utilisent pas cette information spécifique à chaque jeu. Machado et al. [2018] recommandent donc de toujours utiliser l'information de game over pour terminer les épisodes que ce soit durant l'entraînement ou l'évaluation. Cela est plus en accord avec l'idée d'entraîner des algorithmes généraux n'utilisant pas d'information spécifique à la tâche sur laquelle ils sont entraînés.

Ensemble des actions Machado et al. [2018] recommandent de ne pas utiliser *l'ensemble minimal d'actions utiles* spécifique à chaque jeu Atari, i.e. les actions qui ont véritablement un effet sur le jeu Atari en question. Ils proposent de toujours utiliser les 18 actions possibles sur la console Atari. Cela retire encore des informations spécifiques à chaque jeu et réduit la complexité pour reproduire les résultats. En effet, pour certains jeux, cet ensemble minimal d'actions utiles est différent d'une version à une autre de la librairie standard Atari : par exemple un problème pour reproduire les résultats publiés sur le jeu Breakout provenait de cela [Graetz, 2018].

Rapporter les résultats Comme relevé par Machado et al. [2018], rapporter les résultats du meilleur agent au cours de l'apprentissage induit un biais statistique et est malheureusement la méthode la plus souvent utilisée pour rapporter les résultats sur les jeux Atari [Mnih et al., 2015, Hessel et al., 2018, Dabney et al., 2018]. De plus, dans la majorité des cas, il n'est pas dit à quel moment de l'apprentissage ce meilleur agent est obtenu et ainsi cela ne donne aucune information sur la stabilité et l'efficacité de l'entraînement. Ils proposent donc plutôt de rapporter les scores à des moments précis et fixes de l'apprentissage en moyennant les scores obtenus sur k épisodes consécutifs (on a pris $k = 100$). Cela permet de donner une information sur la stabilité de l'entraînement tout en enlevant le biais induit lorsque le résultat du meilleur agent est rapporté.

Ces quatre recommandations de Machado et al. [2018] sont un premier pas vers une standardisation des procédés d'évaluation des agents sur le benchmark Atari et ont été toutes les quatre intégrées à SABER comme nous l'expliciterons dans la section 4.3.

Pour conclure, il y a récemment de manière générale une sorte de crise dans l'ensemble du domaine de l'apprentissage automatique et particulièrement avec les réseaux de neurones et l'apprentissage par renforcement. Ainsi dans la suite de l'article de Henderson et al. [2018], J. Pineau a introduit une *Machine Learning reproducibility checklist* [Pineau, 2019] permettant d'effectuer des comparaisons équitables et de s'assurer de la reproductibilité des travaux. Cette liste a depuis été ré-utilisée dans de nombreuses conférences pour inciter les chercheurs à rendre leurs travaux plus simples à reproduire et à comparer. La conférence ICLR a aussi créé un concours de reproductibilité (le *ICLR Reproducibility Challenge*) qui en est maintenant à sa 3ème édition.

4.2.2 Apprentissage par renforcement profond fondé sur la fonction de valeur

Comme nous l'avons vu dans le Chapitre 2, l'algorithme DQN [Mnih et al., 2015] est l'un des premiers algorithmes de DRL évalué sur l'ensemble des jeux Atari avec exactement les mêmes hyperparamètres et architecture de réseau de neurones. Depuis ce succès, DQN a été amélioré et étendu pour devenir plus robuste, plus efficace et atteindre de meilleures performances.

Rainbow [Hessel et al., 2018] Rainbow est la combinaison en un seul algorithme de six de ces améliorations : Fortunato et al. [2017], Van Hasselt et al. [2016], Schaul et al. [2015], Bellemare et al. [2017], Wang et al. [2016], Mnih et al. [2016]. Des études d'ablations de Rainbow, c.f. Figure 4.2, ont permis de déterminer que les deux améliorations les plus importantes était Prioritized Experience Replay (PER) [Schaul et al., 2015] et C51 [Bellemare et al., 2017].

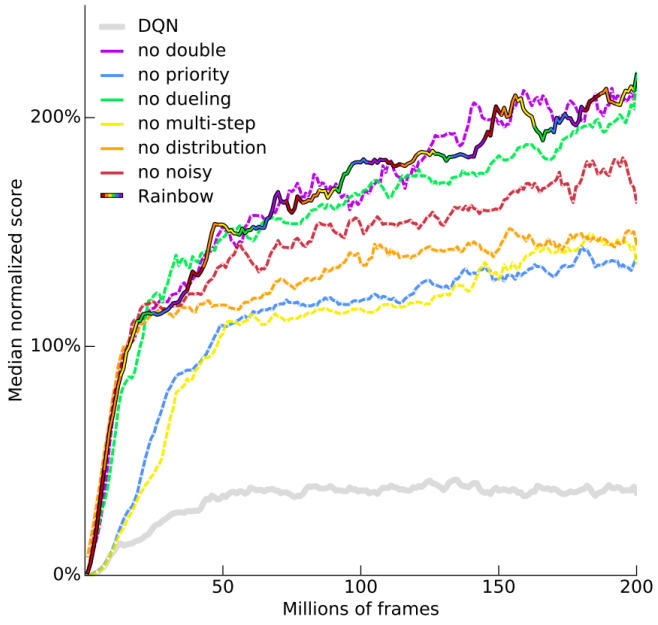


FIGURE 4.2 – Études d'ablation sur l'algorithme Rainbow. L'idée est de retirer chaque composant un par un et de comparer les performances finales. Cela permet de montrer que les deux articles les plus importants sont Prioritized Experience Replay (expérience *no priority*) [Schaul et al., 2015] et C51 (expérience *no distribution*) [Bellemare et al., 2017] (RL distributionnel). Image provenant de [Hessel et al., 2018]

L'idée dans l'article PER [Schaul et al., 2015] est de sélectionner les expériences de la mémoire de reprise proportionellement à leur *surprise*, i.e. plus l'erreur de prédiction des Q-valeurs est grande, plus on l'échantillonne pour faire la rétropropagation. L'intuition étant de plus apprendre des situations difficiles et de laisser de côté les expériences où

l'agent arrive déjà à prédire les Q-valeurs.

C51 [Bellemare et al., 2017] est le premier algorithme de *renforcement distributionnel* (*Distributional RL* en anglais) qui est le domaine du renforcement où l'agent estime la distribution entière de la fonction de valeur Q et non plus seulement la moyenne de cette fonction. Ce qui est très intéressant est que même si l'on choisit toujours les actions en moyennant la fonction Q, i.e. on n'utilise jamais véritablement la distribution totale de cette fonction pour agir, estimer la distribution entière agit comme une tâche auxiliaire qui renforce le signal d'apprentissage du renforcement et permet d'améliorer considérablement l'efficacité d'apprentissage ainsi que les performances finales de l'agent.

Les autres articles utilisés dans l'algorithme Rainbow sont par ordre d'importance n-step DQN [Mnih et al., 2016], Noisy Network [Fortunato et al., 2017], Double DQN [Van Hasselt et al., 2016] et dueling Network [Wang et al., 2016]. L'idée de n-step DQN est extrêmement simple et a déjà été introduite précédemment, Chapitre 2 Section 2.4.2, et consiste à faire un boostrap à n-step. Cependant, il est très intéressant de noter que cette amélioration a un impact très fort sur les performances, supérieur à toutes les approches qui vont suivre. Dans l'article Noisy Network [Fortunato et al., 2017], les auteurs ont remplacé les couches *fully connected* par des *noisy fully connected*. Ces couches *noisy fully connected* prennent en entrée les features précédentes ainsi qu'un vecteur de bruit, l'idée est qu'ainsi l'agent peut apprendre de lui même son taux d'exploration. En effet, au début de l'entraînement par exemple, l'agent va apprendre à utiliser principalement le bruit dans l'optique de faire des actions aléatoires pour mieux explorer son environnement. Ensuite, l'agent peut apprendre à réduire l'impact du bruit voire quasiment l'annihiler si l'exploration n'est plus nécessaire. Concernant les deux derniers composants, Double DQN et dueling Network, les études d'ablations sur Rainbow ont montré que ces deux améliorations n'avait pas d'impact notable sur les performances, nous verrons même que nous avons retiré le dueling Network de notre architecture pour l'application à la conduite autonome dans le chapitre suivant, Chapitre 5.

Implicit Quantiles Network (IQN) [Dabney et al., 2018] L'article IQN est une amélioration de l'algorithme C51 et s'intègre lui-aussi dans la ligne des articles sur le renforcement distributionnel. Les résultats de cet article sont particulièrement remarquables. En effet, sur le benchmark Atari, IQN atteint quasiment les mêmes performances que Rainbow avec ses six composants. Dans IQN la distribution de la fonction Q est estimée implicitement avec des quantiles, alors que la fonction Q est estimée avec une distribution en catégorie dans C51 dont le support est fixe et seule la masse dans chaque intervalle est apprise. Les quantiles dans IQN sont tirés selon une loi aléatoire (généralement une loi uniforme $U([0,1])$) et forment ainsi une distribution implicite, i.e. le réseau doit estimer la valeur du quantile tiré aléatoirement de la fonction Q ; c'est de là que vient le terme *Implicit* de IQN. La plus grande contribution de IQN est théorique, ils ont effectivement réussi à réellement minimiser la distance de Wasserstein entre l'estimée actuelle de la fonction Q et la vraie fonction Q. Le premier algorithme de renforcement distributionnel, C51 [Bellemare et al., 2017], passait par une approximation avec une minimisation de la KL-divergence. Expliquer plus en détail pourquoi cela apporte une

véritable amélioration est en-dehors de la portée de ce manuscrit.

Distributed PER (Ape-X) [Horgan et al., 2018] Il est important de noter que tous les algorithmes décrits précédemment sont *mono-agent*, i.e. il y a un seul agent qui interagit avec une seule instance de l'environnement et cet agent va aussi effectuer l'apprentissage et mettre à jour les poids du réseau de neurones. Horgan et al. [2018] ont proposé une architecture distribuée pour appliquer du DRL fondé sur la fonction de valeur à grande échelle. Cette architecture permet de générer une quantité de données pour l'apprentissage bien supérieure aux approches mono-agent. Cet algorithme sépare l'apprentissage de l'interaction avec l'environnement : les acteurs interagissent avec leur propre environnement en choisissant les actions via un réseau de neurones partagé entre tous les acteurs et accumulent leurs expériences dans une mémoire de reprise partagée. L'*agent d'apprentissage* (en anglais *learner*) quant à lui échantillonne les expériences de la mémoire de reprise partagée et met à jour les poids du réseau de neurones par rétropropagation. L'agent d'apprentissage ne va ainsi jamais interagir avec l'environnement et lui seul fait réellement l'apprentissage, i.e. la rétropropagation. Initialement, Horgan et al. [2018] ont appliqué leur architecture distribuée à PER [Schaul et al., 2015], c'est à dire que l'agent d'apprentissage échantillonne les expériences en fonction de leur surprise comme expliqué précédemment. Leur architecture distribuée est illustrée en figure 4.3.

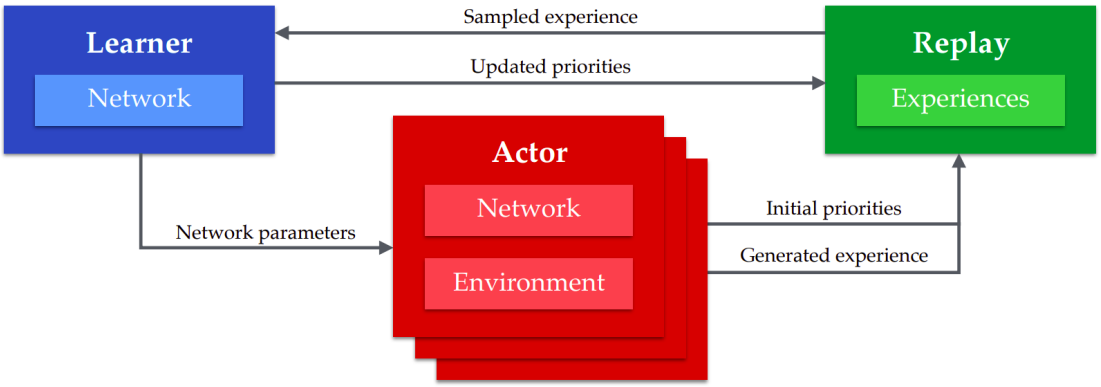


FIGURE 4.3 – Architecture de Ape-X [Horgan et al., 2018] : une multitude d'acteurs, chacun interagissant avec sa propre instance de l'environnement, génèrent des expériences et les rajoutent à une mémoire de reprise partagée tout en calculant leur priorité initiale proportionnelle à la surprise, i.e. l'erreur entre la Q-valeur estimée et la Q-valeur cible. L'agent d'apprentissage, unique, échantillonne des expériences de cette mémoire partagée selon leur priorité et met à jour les poids du réseau. Les réseaux des acteurs sont périodiquement mis à jour avec les poids les plus récents de l'agent d'apprentissage. Image provenant de [Horgan et al., 2018]

Cependant, comme nous le verrons prochainement, cette architecture distribuée peut s'appliquer à n'importe quel algorithme mono-agent fondé sur la fonction de valeur comme Rainbow ou IQN.

4.3 SABER : un benchmark Atari standardisé

4.3.1 Temps maximal d'un épisode

Un paramètre majeur n'est pas pris en compte dans le travail de Machado et al. [2018] : le temps maximal d'un épisode. Ce paramètre termine les épisodes après un temps fixé même si le jeu n'est pas réellement fini. Dans la majorité des travaux récents sur Atari [Dabney et al., 2018, Horgan et al., 2018, Hessel et al., 2018], le temps de jeu maximal utilisé est 30 minutes et seulement 5 minutes dans l'article de Machado et al. [2018]. Cela signifie que les scores rapportés ne peuvent pas être comparés de manière équitable. Par exemple, sur les jeux Atari simples (comme Atlantis ou Enduro), l'agent ne meurt jamais et le score est plus ou moins proportionnel au temps de jeu alloué : le score rapporté sera donc six fois plus grand si on limite à 30 minutes plutôt qu'à 5 minutes.

De plus, nous argumentons que ce temps maximal peut rendre les comparaisons non significatives. Sur plusieurs jeux (comme Video Pinball ou Atlantis) les scores rapportés par Ape-X [Horgan et al., 2018], Rainbow [Hessel et al., 2018] et IQN [Dabney et al., 2018] sont pratiquement identiques. Cela est dû au fait que les agents atteignent la limite de temps alloué sans mourir et obtiennent le plus grand score possible en 30 minutes : les différences en score ne sont dues qu'à des variations mineures et non pas à des véritables différences d'algorithmes. Cela a pour conséquence que plus les agents sont bons, plus il y a de jeux qui ne sont plus comparables car les agents atteignent le meilleur score possible dans la limite de temps imparti.

Ce paramètre est aussi une source d'ambiguïté et d'erreur. Le meilleur score sur Atlantis (2,311,815) est rapporté par l'algorithme PPO [Schulman et al., 2017] mais ce score est certainement erroné car il n'est pas atteignable en seulement 30 minutes ! Le premier travail de renforcement distributionnel [Bellemare et al., 2017] a fait lui aussi cette erreur et les auteurs ont rapporté des résultats erronés avant de rajouter un erratum dans une version suivante sur Arxiv.

En fait, nous pensons que le temps des épisodes ne devrait pas être limité. L'article initial sur ALE [Bellemare et al., 2013] indiquait que ce paramètre (de temps maximal) est nécessaire pour s'assurer que les jeux terminent toujours. En effet sur certains jeux Atari particulièrement difficiles, comme Pitfall et Tennis, une exploration aléatoire apporte bien plus de récompenses négatives que de récompenses positives : l'agent finit donc par apprendre à ne rien faire, par exemple à ne jamais servir dans le jeu Tennis. Pour cette raison, nous suggérons d'ajouter un *temps bloqué maximum*, i.e. un temps terminant l'épisode si l'agent est bloqué pendant trop longtemps. Cela permet de s'assurer que les épisodes terminent même quand l'agent a appris à ne rien faire tout en enlevant l'inconvénient d'ajouter une limite de temps qui détériore l'évaluation sur tous les autres jeux. Finalement, les records du monde humains sur les jeux Atari ont été effectués après plusieurs heures de jeu et auraient été inatteignables en seulement 30 minutes.

Pour toutes ces raisons, nous recommandons de laisser le temps maximum d'un épisode illimité (en pratique, on a utilisé une limite de 100 heures). De plus, nous suggérons d'ajouter un *temps bloqué maximum*. Cette astuce permet de contourner certains bugs

dans l'environnement ALE (dans certains jeux, le jeu bugue et s'arrête indéfiniment, il faut donc détecter cela pour redémarrer l'épisode) tout en mettant tous les scores rapportés sur une même base rendant possible une comparaison avec les records du monde. En pratique, nous avons utilisé un temps bloqué maximum de 5 minutes, c'est à dire que l'épisode se termine si l'agent n'a reçu aucune récompense sur les 5 dernières minutes. Nous avons testé avec un temps bloqué maximum de 30 minutes mais cela n'amenait aucune différence et rendait l'évaluation des agents plus longues. La raison pour cela est que les jeux Atari sont censés donner des récompenses assez régulièrement (l'idée est de guider le joueur humain dans le jeu) et un agent bloqué 5 minutes sera en fait aussi bloqué 30 minutes...

4.3.2 La référence des records du monde humains

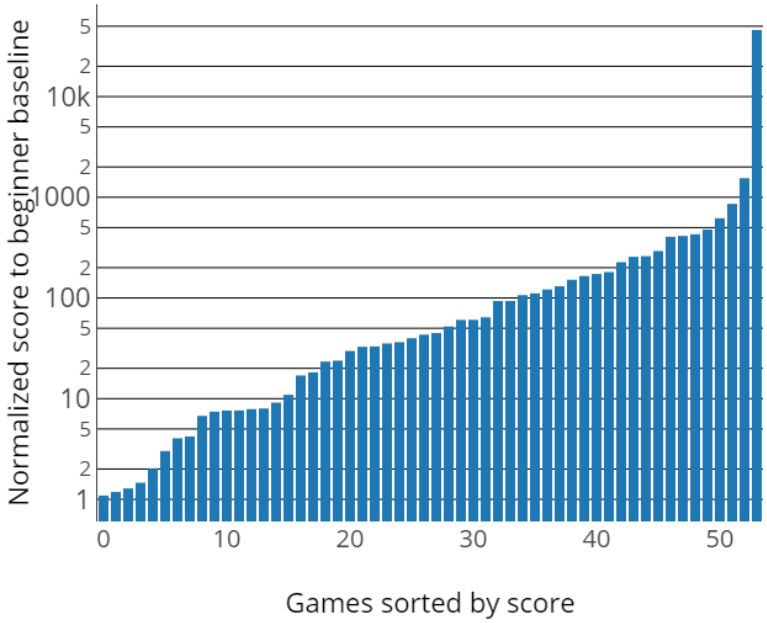


FIGURE 4.4 – Les records du monde humains comparés à la référence humaine introduite par [Mnih et al., 2015] (en échelle log). On constate donc que sur la grande majorité des jeux, le record du monde est plus de 100 fois supérieur à la référence humaine utilisée précédemment !

Une manière très courante d'évaluer les intelligences artificielles dans les jeux est de laisser les agents concourir avec les champions du monde humains. Récemment, de nombreux exemples de cela se sont produits pour des agents appris par renforcement : Alphago [Silver et al., 2016] contre Lee Sedol pour le go, AlphaStar [Vinyals et al., 2019] contre Mana pour StarCraft2 ou OpenAI Five [OpenAI, 2018] contre des professionnels sur Dota 2. Dans le même esprit, une des métriques les plus utilisées pour évaluer les agents sur Atari est de les comparer à la référence humaine introduite par Mnih et al.

[2015]. Les travaux précédents utilisent le score humain normalisé, c'est à dire que 0% est le score d'un agent agissant aléatoirement et 100% est le score de la référence humaine sur ce jeu. Cela permet de rassembler les performances sur tous les jeux Atari en un seul nombre au lieu de comparer individuellement les scores bruts sur chacun des 61 jeux Atari. Cependant, nous argumentons que cette référence humaine est très loin d'être représentative du meilleur joueur humain, ce qui signifie que l'utiliser pour affirmer des performances surhumaines est totalement erroné. En effet, sur certains jeux, le score de cette référence humaine est relativement faible et facile à battre après seulement quelques parties. Les records du monde sont disponibles pour 58 des 61 jeux Atari évalués [1]. Évaluer ces records du monde en utilisant le score humain normalisé usuel (i.e. avec la référence humaine de Mnih et al. [2015]) a une médiane de 4 400% et une moyenne de 99 300% (voir Figure 4.4 pour les détails sur chaque jeu). A titre de comparaison, les performances de l'état de l'art atteint par l'algorithme Rainbow [Hessel et al., 2018] ne sont que 200% de médiane (22 fois moins) et 800% de moyenne (plus de 100 fois moins !). Ainsi, nous argumentons qu'utiliser le score humain normalisé avec les records du monde donne une bien meilleure indication des performances des agents et de la marge d'amélioration restante.

4.3.3 Description de SABER

Dans cette section, nous introduisons SABER : un ensemble de procédés d'entraînement et d'évaluation sur le benchmark Atari permettant de faire des comparaisons équitables et d'être reproductible. De plus, ces procédures permettent de se comparer avec la référence des records du monde introduite dans la section précédente. Cela permet d'avoir une idée précise de la marge d'amélioration entre les agents généraux et les meilleurs joueurs humains.

Procédures d'évaluation et d'entraînement Toutes les recommandations introduites dans les sections précédentes sont rassemblées dans la Table 4.1 et constituent le standard SABER. Il est important de noter que ces recommandations doivent être utilisées à la fois durant l'entraînement et l'évaluation des agents. Le travail récent Go-Explore [Ecoffet et al., 2019] a ouvert un débat sur le fait qu'on puisse enlever ou non la stochasticité durant l'entraînement. Ils ont en effet rapporté des résultats largement supérieurs à l'état de l'art courant sur le jeu Montezuma's Revenge réputé pour être extrêmement difficile et nécessitant une exploration très efficace. Cependant leur méthode n'est en fait applicable que si la stochasticité est retirée durant l'entraînement ce qui n'est pas possible dans la majorité des applications réelles. Ecoffet et al. [2019] ont donc conclu leur travail en indiquant qu'il faut avoir en fait bien rapporté les résultats différemment selon que la stochasticité de l'environnement est enlevée durant l'entraînement ou non. En ce qui nous concerne, nous avons choisi d'utiliser les mêmes conditions durant l'entraînement et l'évaluation car cela est bien plus en ligne avec les applications réelles.

1. sur le site web de TwinGalaxies `https://www.twingalaxies.com/games.php?platformid=5`

Paramètre	Valeur
Sticky actions	$\xi = 0.25$
Information du nombre de vies	Pas autorisée
Ensemble d'action	18 actions
Temps bloqué maximal	5 min (18000 images)
Temps maximal d'un épisode	Infini (100 heures)
État initial et graine aléatoire	Même état initial et graines qui varient

TABLE 4.1 – Les paramètres à utiliser dans les jeux Atari pour le standard SABER

Rapporter les résultats En accord avec les recommandations de Machado et al. [2018], nous suggérons de rapporter la moyenne des scores sur 100 épisodes consécutifs à certains moments spécifiques, ici 10M, 50M, 100M et 200M images d'entraînement. Cela retire le biais induit lorsque les scores du meilleur agent obtenu durant l'entraînement sont rapportés et permet de comparer les performances avec plus ou moins de données d'entraînement. Comme dans les travaux précédents, nous rapportons les performances de l'agent par la moyenne et la médiane des scores normalisés par rapport à la référence humaine, à la différence qu'on utilise la référence des records humains. La médiane est en fait bien plus représentative : la moyenne est grandement influencée par certains cas particuliers comme les rares jeux où la performance est surhumaine. Pour les jeux où l'agent ne meurt jamais et obtient un score infini, le score est articiellement mis à 200% par rapport à la référence humaine. Nous proposons aussi de classer les jeux selon leur difficulté et de les représenter dans un histogramme. Nous définissons cinq classes : *échec* (<1%), *mauvais (<10%)*, *moyen (<50%)*, *bon (<100%)*, *surhumain* (>100%). Même si ces valeurs sont arbitraires, elles permettent d'avoir une idée plus précise de la distribution de la difficulté des jeux que la seule médiane. Nous allons en particulier voir que la grande majorité des jeux sont en fait dans les classes *échec* et *mauvais*, ce qui montre le très grand fossé qui reste à franchir entre les performances des agents et les records humains.

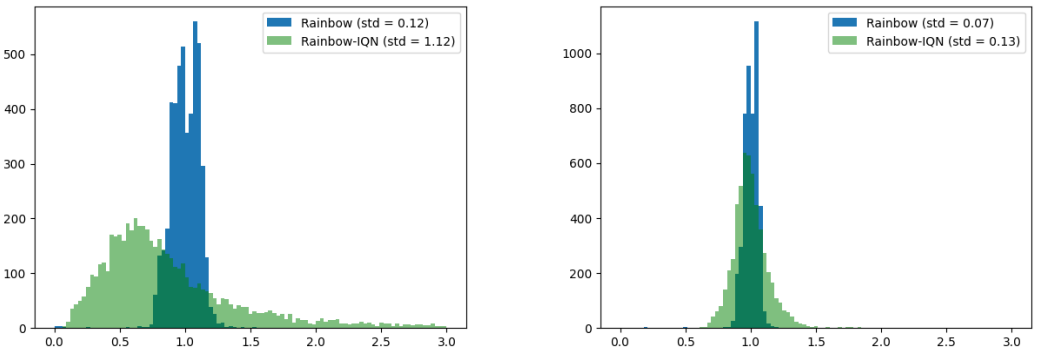
4.4 Rainbow-IQN Ape-X

Dans cette section, nous décrirons plus précisément comment nous avons développé notre algorithme, en particulier le réglage du paramètre ω (*l'exposant de priorité*) qui a un impact sur l'échantillonnage des expériences par l'agent d'apprentissage. Dans un deuxième temps, nous détaillerons notre implémentation pratique disponible en open-source [2] permettant de reproduire facilement nos résultats.

Problème avec les priorités des expériences Comme mentionné précédemment, nous avons combiné l'algorithme Rainbow [Hessel et al., 2018], qui atteignait les meilleures

2. https://github.com/valeoai/rainbow-iqn-apex

performances sur Atari, avec IQN [Dabney et al., 2018]. L'idée est que IQN est une pure amélioration de l'algorithme C51 [Bellemare et al., 2017] qui est l'une des parties de Rainbow. La combinaison de ces deux algorithmes était donc naturelle et présageait une bonne amélioration, car IQN seul atteint déjà des performances quasi comparables au Rainbow complet. Cependant, après avoir effectué cette combinaison, les tests préliminaires ont eu des résultats très décevants : cela était dû aux hyperparamètres initiaux de Rainbow qui n'était pas adaptés à IQN. En effet, les expériences sont échantillonnées proportionnellement à leur surprise à la puissance de l'hyperparamètre ω (*l'exposant de priorité*). Or, lorsque l'on a comparé la distribution des surprises obtenues par l'algorithme Rainbow et par Rainbow-IQN, nous avons découvert que dans le deuxième cas, la distribution était beaucoup plus étalée comme illustrée sur la Figure 4.5.



(a) Distribution originelle des surprises

(b) Distribution des surprises après appliqué l'exposant de priorité ($\omega_{Rainbow} = 0.5$ et $\omega_{IQN} = 0.2$)

FIGURE 4.5 – Distribution des surprises de Rainbow et Rainbow-IQN : distributions originelles à gauche et celle obtenues après application de l'exposant de priorité à droite avec $\omega_{Rainbow} = 0.5$ et $\omega_{IQN} = 0.2$. On constate que la distribution originelle de Rainbow-IQN est bien plus étalée ce qui rendait l'entraînement instable car certaines transitions étaient trop échantillonnées et d'autres jamais. En fait, même en prenant un exposant de priorité bien inférieur (0.2 au lieu de 0.5) la déviation standard de la distribution de Rainbow-IQN est supérieure à celle de Rainbow seul mais en bien moindre mesure.

Ainsi, lors de l'entraînement avec Rainbow-IQN, certaines transitions était beaucoup trop échantillonnées (et d'autres jamais) ce qui rendait l'entraînement instable. D'ailleurs ce paramètre avait déjà été problématique pour l'algorithme Rainbow et avait dû être re-réglé. Pour résoudre ce problème, nous avons testé 4 valeurs différentes pour ω sur 5 jeux Atari : 0.1, 0.15, 0.2, 0.25 au lieu de 0.5 initialement dans Rainbow, avec 0.2 qui donne les meilleures performances. Les 5 jeux qui ont été utilisés sont Alien, Battle Zone, Chopper Command, Gopher et Space Invaders. Ces jeux ont été choisis car ils étaient représentatifs du niveau de difficulté moyen des jeux Atari, i.e. ni trop simple, ni trop difficile, ce qui permet d'avoir une comparaison plus significative des performances des

différentes expériences.

Implémentation pratique En pratique, nous sommes partis de l'implémentation open-source de Rainbow en PyTorch [Paszke et al., 2019] de Kaikhin [Kaixhin, 2018]. Nous avons ensuite vérifié les performances de cette implémentation initiale avec les exactes conditions et hyperparamètres de Rainbow pour s'assurer que nos résultats étaient cohérents. Après cette vérification, nous avons implémenté une version distribuée de Rainbow en suivant l'article Ape-X [Horgan et al., 2018]. Pour la mémoire de reprise partagée, nous avons utilisé une base de donnée clé-valeur avec REDIS [3]. La mémoire de reprise est gardée en RAM pour que les accès mémoires soient les plus rapides possible. C'est cette implémentation distribuée qui nous a pris le plus de temps, en effet certaines optimisations n'étaient pas triviales à implémenter sur une architecture distribuée. En particulier, l'échantillonnage des expériences selon leur priorité doit utiliser un *SumTree* afin d'être efficace : complexité en $O(log(n))$ au lieu de $O(n)$ pour un échantillonage naïf avec n le nombre d'expériences enregistrées dans la mémoire. L'optimisation de ce SumTree a pris du temps, il a fallu paralléliser le plus possible les appels à la mémoire REDIS pour éviter de perdre trop de temps en lecture. Nous avons aussi dû mettre en place des systèmes de lock pour éviter des accès concurrentiels aux mêmes données. Pour donner un ordre d'idée, cette implémentation distribuée a nécessité plus d'un mois de développement et comporte plus de 3000 lignes de code à comparer aux 900 lignes de code de l'implémentation initial de Rainbow de Kaixhin [Kaixhin, 2018]. Nous argumentons que notre implémentation distribuée est une amélioration fondamentale comparée aux implémentations *mono-agent* comme celle de Dopamine [Castro et al., 2018] ou de Kaikhin [Kaixhin, 2018]. En effet, cela permet d'appliquer notre algorithme sur des environnements où la génération de données est beaucoup plus coûteuse que sur Atari. Nous verrons dans le chapitre suivant, Chapitre 5, que cela nous a ainsi permis d'appliquer notre algorithme sur le simulateur CARLA en utilisant plusieurs instances de CARLA distribuées sur plusieurs machines. Sans notre implémentation distribuée, cela n'aurait pas été possible car le temps de génération des données aurait été extrêmement limitant.

Nous avons ensuite combiné notre implémentation de Rainbow Ape-X avec IQN en partant de l'implémentation en TensorFlow [Abadi et al., 2016] de Dopamine [Castro et al., 2018] pour obtenir notre implémentation finale de Rainbow-IQN Ape-X. Tous les hyperparamètres (sauf *l'exposant de priorité* ω) correspondent exactement à ceux rapportés dans Rainbow et dans IQN.

4.5 Expériences

Dans cette section, nous allons décrire nos expériences effectuées en utilisant le standard SABER. Pour tous les paramètres qui ne sont pas mentionnées dans SABER (comme l'architecture du réseau, le prétraitement des images etc) nous avons soigneusement suivi les paramètres utilisés dans les articles de Rainbow et IQN. Il est important

3. https://redis.io/

de noter qu'afin de faire les comparaisons les plus équitables possibles, nous n'avons pas utilisé notre version distribuée dans nos expériences, i.e. nous évaluons Rainbow et Rainbow-IQN avec un unique agent. L'intérêt de la version distribuée sera surtout démontré dans le chapitre suivant pour appliquer du DRL à un environnement bien plus complexe où la génération de données est coûteuse. L'entraînement d'un agent sur un jeu Atari nécessite environ une semaine ce qui donne environ une année-GPU pour l'entraînement sur l'ensemble du benchmark Atari. Pour cette raison, nous n'avons pu entraîner nos agents qu'avec une seule graine aléatoire sur l'ensemble des jeux Atari. La durée combinée de toutes les expériences effectuées représente plus de 2 années-GPU. Les agents sont entraînés avec le standard SABER sur les 61 jeux Atari et évalués avec la référence des records du monde sur 58 jeux. Les scores à 5 minutes et à 30 minutes de jeu sont gardés durant l'entraînement pour pouvoir se comparer aux travaux précédents.

4.5.1 Infrastructure d'entraînement : Utilisation d'un centre de calcul à distance, le CCRT

Comme indiqué ci-dessus, les expériences ont nécessité plus de 2 années-GPU. Les ressources de calculs disponibles à Valeo et aux Mines étaient largement insuffisantes pour atteindre un tel nombre d'expériences en un temps raisonnable. C'est pour cette raison que nous avons utilisé les ressources d'un centre de calcul très important, le Centre de Calcul Recherche et Technologie (CCRT) du CEA. L'utilisation de ce centre de calcul nous a en particulier permis d'accéder à 4 cartes graphiques Nvidia P100 ainsi qu'à 8 Nvidia V100 ce qui a globalement triplé notre capacité de calcul cumulée entre Valeo et les Mines. De plus, ces ressources nous ont été attribuées gratuitement, car c'était la première fois que de véritables expériences GPU étaient lancées au CCRT. Cela a donc aussi été très formateur et m'a permis de prendre en main un système de partage de ressources à distance, ainsi que de lancer automatiquement de nombreuses expériences sur de nombreux serveurs tout en aidant et en repérant des potentiels manques et erreurs dans la configuration des serveurs du centre de calcul.

4.5.2 Évaluation de Rainbow avec SABER

Algorithme	Rainbow Originel [Hessel et al., 2018]			Rainbow avec Machado et al. [2018]		
	Médiane	Moyenne	Surhumain	Médiane	Moyenne	Surhumain
Performance	4.20%	24.10%	2	2.61%	17.09%	1

Table 4.2 – Moyenne et médiane du score normalisé avec la référence humaine des records du monde. Le nombre d'agents atteignant une performance surhumaine est indiqué. Les scores proviennent de l'article initial de Rainbow [Hessel et al., 2018] et de notre ré-évaluation de Rainbow en suivant les recommandations de Machado et al. [2018] (épisodes limités à 30 minutes de jeu et scores à 200M images d'entraînement).

Ré-évaluer Rainbow permet de mesurer l'impact des recommandations de Machado

et al. [2018] : les sticky actions, ne pas utiliser l'information spécifique à chaque jeu du nombre de vies ainsi qu'utiliser toujours le même ensemble d'action. La Table 4.2 compare les résultats originels rapportés de Rainbow avec les résultats d'un entraînement suivant les recommandations de Machado et al. [2018]. Les scores sont normalisés avec la référence des records humains, avec une durée d'évaluation de 30 minutes à 200M de frames pour être aussi proche que possible des conditions du Rainbow initial pour faire une comparaison équitable. L'impact des procédures d'entraînement standardisées est majeur : comme montré dans le paragraphe suivant, la différence en médiane (1.59%) est comparable avec la différence de performance entre DQN et Rainbow (1.8%, cf Figure 4.8) lorsque les deux sont entraînés sous les mêmes conditions. Cela démontre bien l'importance de procédures d'entraînement et d'évaluation explicites et standardisées.

4.5.3 Évaluation de Rainbow-IQN avec SABER

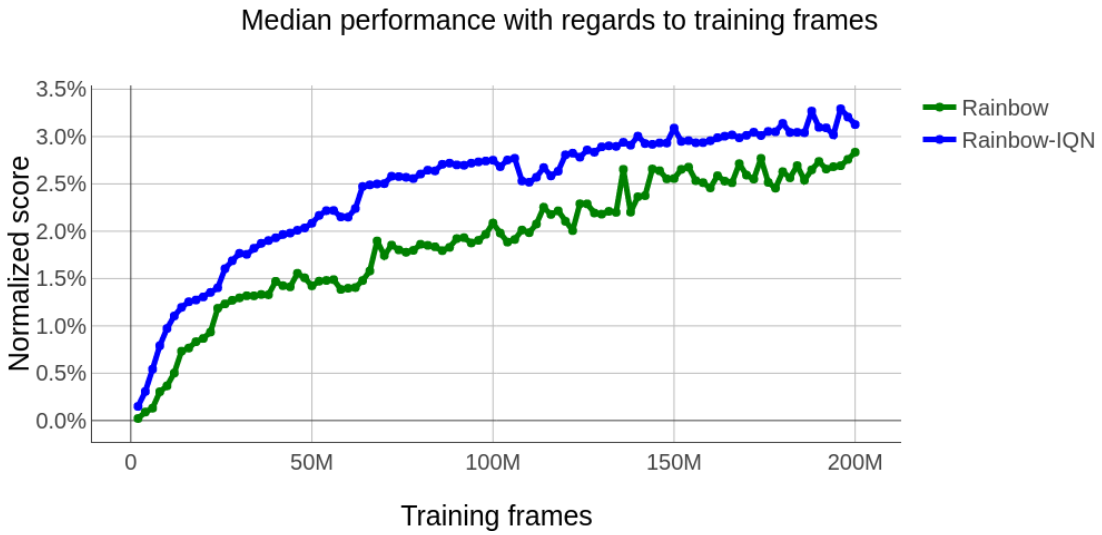


FIGURE 4.6 – Comparaison de Rainbow avec Rainbow-IQN avec le standard SABER : Médiane des scores normalisés en fonction du nombre d'images d'entraînement.

Nous avons comparé les performances de Rainbow et Rainbow-IQN en utilisant la médiane des scores normalisés tout au long de l'entraînement sur la Figure 4.6. Cette figure montre que les performances de Rainbow-IQN sont continuellement meilleures que les performances de Rainbow au long de l'apprentissage. Cependant la majorité des agents, que ce soit Rainbow ou Rainbow-IQN, sont dans les catégories *échec* ou *mauvais* (c.f. Figure 4.7) montrant ainsi le fossé qui reste à franchir avant de réussir à atteindre des performances réellement surhumaines sur le benchmark Atari.

La Figure 4.8 fournit une comparaison entre DQN, Rainbow et Rainbow-IQN. Le temps d'évaluation est mis à 5 minutes pour être cohérents avec les résultats de DQN rapportés par Machado et al. [2018]. Comme prévu, les performances de DQN sont lar-

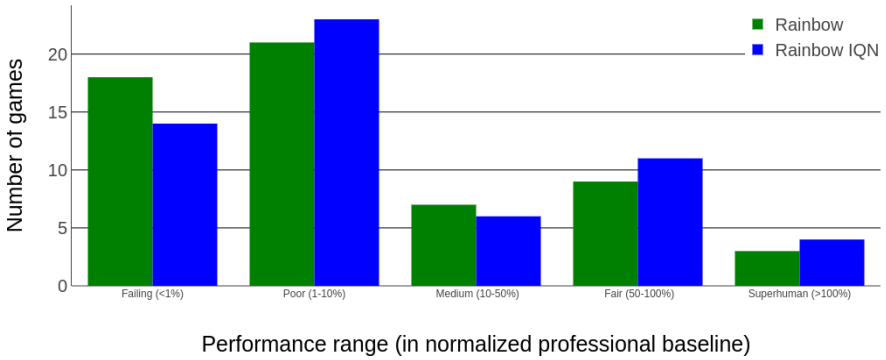


FIGURE 4.7 – Comparaison de Rainbow avec Rainbow-IQN avec le standard SABER : classification de la performance des agents comparée à la référence des records humains (scores pris à 200M images d'entraînement).

gement inférieures pour tous les pas d'entraînement. Comme mentionné précédemment, la différence entre DQN et Rainbow est du même ordre que la différence provenant de procédures d'entraînement différentes, montrant encore une fois la nécessité d'une standardisation.

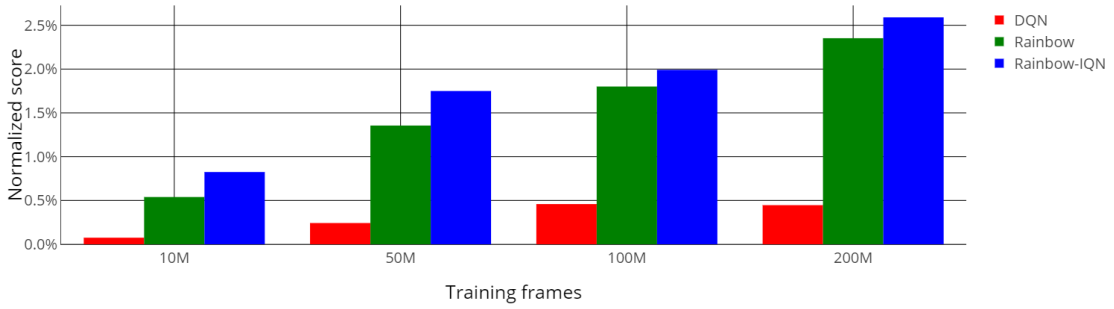


FIGURE 4.8 – Comparaison de la médiane des scores normalisés pour DQN, Rainbow et Rainbow-IQN en fonction du nombre d'images d'entraînement. Le temps maximale d'un épisode est mis à 5 minutes pour permettre une comparaison avec les résultats de DQN rapportés par Machado et al. [2018].

4.6 Conclusion

Dans ce travail, nous avons confirmé l'impact de procédures standards pour entraîner par renforcement et évaluer des agents sur le benchmark Atari. Nous avons aussi introduit un standard consolidé, SABER. L'importance du temps de jeu est souligné : les agents devraient être entraînés et évalués sans aucune limitation en temps de jeu. Pour obtenir une comparaison plus significative, nous avons introduit une nouvelle référence humaine : les records du monde. En utilisant cette nouvelle référence humaine, nous avons

montré que les agents actuels sont en fait très loin d'atteindre la performance des records humains. Nous espérons que ce résultat permettra de renouveler l'intérêt des chercheurs pour le benchmark Atari qui est récemment devenu beaucoup moins populaire. En effet, nous pensons que la perte d'intérêt pour les jeux Atari est en particulier due au fait que les gens présument à tort que ce benchmark est résolu, et que les agents entraînés par renforcement sont déjà surhumains. Finalement, nous avons introduit Rainbow-IQN, un algorithme plus performant que l'état de l'art au temps de publication.

4.6.1 Pourquoi le renforcement est-il si mauvais sur les jeux Atari ?

L'information principale de nos résultats est que les algorithmes de renforcement généraux (i.e. qui n'utilisent aucune information spécifique de l'environnement) sont extrêmement loin des performances des meilleurs joueurs humains. La médiane des scores normalisés de Rainbow-IQN est de 3.1%, ce qui signifie que pour la moitié des agents, la performance de cet agent est à seulement 3% du chemin entre des actions aléatoires et la meilleure performance humaine. Il y a différentes raisons à ces échecs, que nous allons brièvement discuter ici pour donner une intuition générale sur les limitations actuelles des algorithmes de renforcement généraux.

Récompenses tronquées Les récompenses dans les différents jeux Atari ont des ordres de grandeur très variés : de l'ordre de l'unité dans le jeu Pong à presque un million dans le jeu Atlantis. Pour éviter d'avoir des problèmes de gradients qui explosent dans certains jeux ou quasiment nuls dans d'autres (le gradient est multiplié par la récompense), la grande majorité des travaux sur le benchmark Atari tronquent les récompenses entre -1 et 1. Cependant, dans certains jeux, la stratégie optimale pour l'algorithme de renforcement devient différente de celle du joueur humain. En effet, les agents préfèrent donc obtenir plusieurs petites récompenses qu'une seule grande (qui sera du coup tronquée). Ce problème est particulièrement bien représenté dans le jeu Bowling : l'agent apprend à éviter de faire des spares ou des strikes. En effet, la stratégie optimale est de faire 10 strikes d'affilée ce qui apporte une seule grande récompense de 300 (tronquée à +1 pour l'agent) mais la stratégie optimale pour l'agent est de faire tomber les quilles une par une. Cela montre la nécessité de mieux gérer des récompenses de différents ordres de grandeur, par exemple en utilisant une fonction de valeur inversible comme suggéré par Pohlen et al. [2018] ou en utilisant la normalisation Pop-Art [van Hasselt et al., 2016].

Exploration Une autre raison très courante d'échec est un manque d'exploration résultant en un agent bloqué dans un minimum local. Une exploration aléatoire ou les *Noisy Networks* [Fortunato et al., 2017] sont très loin d'être suffisants pour résoudre la plupart des jeux Atari. Dans le jeu Kangaroo par exemple, l'agent apprend à recevoir des récompenses facilement dans le premier niveau mais n'essaye jamais d'aller dans les niveaux suivants. Ce problème peut en plus être amplifié par le tronquage des récompenses : changer de niveau peut apporter une plus grande récompense, mais pour

l'algorithme de renforcement toutes les récompenses positives sont les mêmes (égales à 1). L'exploration est un des domaines les plus étudiés en apprentissage par renforcement, des solutions possibles peuvent passer par de la curiosité [Pathak et al., 2017] ou de l'exploration fondée sur un décompte des états visités [Ostrovski et al., 2017].

Connaissance humaine préalable Les jeux Atari sont conçus pour les joueurs humains, et ils impliquent énormément de connaissances préalables implicites. Cela permet de guider le joueur humain sur des actions probablement positives même si en fait elles n'apportent aucune récompense instantanée (monter sur une échelle, éviter un crane etc). L'exemple le plus représentatif peut se trouver dans le jeu Riverraid : détruire un baril de fuel donne une récompense instantanée mais les collecter permet de jouer plus longtemps. Les agents actuels n'arrivent pas à apprendre cela et ainsi meurent très rapidement par manque de fuel. Même avec une exploration intelligente, cela reste un problème ouvert pour les agents généraux.

Une vidéo que nous avons créée et illustrant quelques-uns de ces problèmes en comparant les styles de jeu des agents appris par renforcement avec notre algorithme Rainbow-IQN et des records humains peut se trouver ici : `https://www.youtube.com/watch?v=oH6P3ksYLek`.

4.6.2 Quid de la voiture autonome ?

Dans ce chapitre, nous avons exclusivement parlé de l'application de l'apprentissage par renforcement aux jeux Atari ce qui pourrait sembler assez éloigné de l'application supposée de cette thèse, la voiture autonome. Mais l'idée de ce chapitre était de développer le meilleur algorithme général, i.e. réussissant à obtenir les meilleures performances possibles sur la grande diversité des jeux Atari en n'utilisant aucune information spécifique. L'idée est que si un algorithme est capable d'apprendre des tâches variées avec les mêmes hyperparamètres et architecture, cet algorithme devrait pouvoir être transférable facilement et être performant sur n'importe quelle autre tâche, comme la conduite autonome. Cet algorithme devait aussi respecter la contrainte indiquée dans le Chapitre 2, être *off-policy* afin d'être plus efficace en donnée. Nous verrons dans le chapitre suivant, comment nous avons appliqué notre algorithme à une tâche très complexe, la conduite en environnement urbain. D'ailleurs, dans les travaux suivants (Chapitre 5 et 7), le caractère général sera totalement retiré, l'idée est de maintenant obtenir les meilleurs résultats possibles sur une unique tâche (mais particulièrement ardue), nous allons donc bien évidemment abuser de certaines informations spécifiques et régler nous-même notre fonction de récompense !

Chapitre 5

RL pour la conduite urbaine en utilisant des *indices implicites*

Les résultats de ce chapitre ont permis de remporter la première place au challenge CARLA 2019 pour la catégorie "Cameras Only" et la deuxième place pour la catégorie "Camera and LIDAR". Nous avons aussi remporté le challenge CARLA en 2020 sur la catégorie "All Sensors".

Les contributions de ce chapitre ont ensuite été publiées dans [Toromanoff et al., 2020] :

Marin Toromanoff, Emilie Wirbel, and Fabien Moutarde. End-to-end model-free reinforcement learning for urban driving using implicit affordances. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7153–7162, 2020.

5.1 Introduction

Dans un premier temps, en section 2.5.3, nous avons expliqué quel type d'algorithmes semble le plus approprié pour être appliqué à la voiture autonome : les algorithmes fondés sur la fonction de valeur. Dans le chapitre précédent, Chapitre 4, nous avons présenté notre algorithme fondé sur la fonction de valeur qui atteint l'état de l'art sur le benchmark Atari : Rainbow-IQN Ape-X [Toromanoff et al., 2019]. Nous avons en particulier expliqué en quoi la partie distribuée (Ape-X [Horgan et al., 2018]) permet d'espérer converger en un temps raisonnable lorsque l'environnement ne génère pas d'expériences suffisamment rapidement. Dans ce chapitre, nous allons maintenant nous intéresser à l'application concrète de cet algorithme sur une tâche complexe : la conduite en environnement urbain.

La conduite urbaine est probablement une des tâches les plus ardues à résoudre pour les voitures autonomes, particulièrement les interactions aux intersections avec les feux tricolores, les piétons pouvant traverser et les autres véhicules naviguant sur les différentes voies. Cette tâche est loin d'être résolue aujourd'hui et il semble compliqué voire même impossible de gérer un problème aussi difficile et variable avec des approches classiques fondées sur des règles. C'est pourquoi récemment, de nombreux travaux s'intéressent à des systèmes de bout-en-bout, i.e apprendre la prise de décision et le contrôle à partir des données capteurs sans se reposer sur des règles faites à la main.

Comme indiqué en introduction de cette thèse, deux possibilités existent afin de faire de l'apprentissage de bout-en-bout, l'apprentissage par imitation et l'apprentissage par renforcement. Le premier essaye d'apprendre une politique à partir de trajectoires expertes. Dans le cas de la voiture autonome, l'apprentissage par imitation essaye donc de reproduire le comportement d'un conducteur humain. Nous ferons une description détaillée de l'apprentissage par imitation dans le chapitre suivant, Chapitre 6, en particulier comme méthode actuellement privilégiée pour appliquer de l'apprentissage de bout-en-bout sur véhicule réel.

La seconde méthode est l'apprentissage par renforcement qui laisse l'algorithme apprendre de lui-même à partir d'une récompense donnée à chaque action prise par l'agent. Un des problèmes majeurs du renforcement est qu'il a généralement besoin d'une quantité de données extrêmement importante pour converger, encore plus conséquente que l'apprentissage supervisé. Une conséquence directe de cette *inefficacité en données* est qu'il est extrêmement ardu de faire converger un réseau de neurones comportant un grand nombre de paramètres. De plus, la plupart des algorithmes de renforcement utilisent une mémoire de reprise qui permet d'apprendre d'expériences passées mais ce type de mémoire peut limiter la taille des entrées à stocker (i.e. la taille des images). C'est pour ces deux raisons que les réseaux de neurones et les tailles d'images utilisés dans les algorithmes de DRL sont généralement très légers comparés à ceux utilisés traditionnellement en apprentissage supervisé. Ils sont ainsi potentiellement moins performants et pas assez expressifs pour résoudre une tâche aussi complexe que la conduite urbaine. C'est pourquoi la majorité des applications de DRL à la voiture autonome se limitent

à des cas plus simples, par exemple le contrôle du volant pour du maintien de voie, ou bien les jeux de courses comme vu en Chapitre 3. Finalement, un dernier problème avec l'apprentissage par renforcement est que l'algorithme apparaît comme une *boîte noire* dont il est difficile de comprendre comment elle prend ses décisions.

Une méthode prometteuse qui permettrait de résoudre le problème de *l'inefficacité en données* ainsi que le problème de la boîte noire est d'utiliser l'estimation d'indices importants pour le choix des actions comme tâche auxiliaire. L'idée est de faire estimer à l'agent des informations qui sont utiles pour apprendre le bon comportement. Pour la conduite urbaine, des exemples d'indices qui semblent particulièrement utiles sont la carte de segmentation sémantique, la distance au centre de la voie, la présence et l'état de feux tricolores etc... Il est important de noter que l'agent n'a pas accès directement à ces indices, mais doit les estimer uniquement à partir des données brutes venant des capteurs. Ces informations estimées par l'agent peuvent ensuite être utilisées de différentes manières, soit par un algorithme de contrôle classique comme dans [Sauer et al., 2018], soit comme coût auxiliaire pour aider le réseau de neurones à trouver des caractéristiques images meilleures pour la tâche finale de conduite dans [Mehta et al., 2018], ou bien dans une approche de renforcement fondé sur un modèle de l'environnement comme dans le travail très récent [Pan et al., 2019]. Ces indices permettent aussi d'avoir un retour sur comment la décision a été prise. Par exemple, si l'agent se trompe et estime que le feu est vert et qu'il ne s'arrête pas, on peut en déduire que la mauvaise décision vient d'un problème de perception de l'état du feu. Néanmoins, ce retour ne permet pas d'expliquer l'ensemble des erreurs que l'agent pourrait commettre. En fait, une véritable explicabilité des décisions des IA est un domaine encore aujourd'hui peu exploré et est au-delà de la portée de cette thèse.

Dans la suite de ce chapitre, nous présenterons notre approche pour l'application de l'apprentissage par renforcement à la conduite de bout-en-bout depuis la vision, ce qui inclut le maintien de voie, la détection de l'état des feux tricolores, l'évitement des autres véhicules et des piétons ainsi que la gestion des intersections avec trafic. Pour accomplir cela, nous avons introduit une nouvelle technique que nous appelons *indices implicites*. L'idée est de diviser l'apprentissage en deux phases distinctes : d'abord une structure encodeur-décodeur est entraînée à estimer des *indices* comme l'état du feu tricolore ou la distance au centre de la voie. Ensuite les *features* issues de la partie encodeur sont utilisées comme état pour l'agent appris par renforcement au lieu des images brutes. De cette manière le signal d'apprentissage du renforcement est utilisé seulement pour apprendre la dernière partie du réseau qui est en pratique largement plus petite que la structure encodeur. De plus, comme on utilise seulement les features issues de l'encodeur, la taille nécessaire pour la mémoire de reprise est largement diminuée. En pratique, dans notre cas d'application, les features de l'encodeur utilisent 20 fois moins de mémoire que les images brutes d'entrée.

Nous avons démontré la capacité de notre méthode en remportant en 2019 le *CARLA Autonomous Driving Challenge*[1] que nous détaillerons dans la suite de ce chapitre, section 5.3. A notre connaissance, ce travail est le premier et encore le seul publié (au

1. https://carlachallenge.org/

moment d'écrire cette thèse) à obtenir par apprentissage par renforcement un agent capable de résoudre une tâche aussi complexe de conduite urbaine, particulièrement sur la gestion des feux tricolores.

Nous résumons nos contributions principales de ce chapitre ci-dessous :

— Le premier agent appris par renforcement capable de conduire depuis la vision dans un environnement urbain complexe contenant à la fois le maintien de voie, l'évitement des autres véhicules et piétons ainsi que la gestion d'intersection avec de la circulation et des feux tricolores

— Introduction d'une nouvelle méthode, qu'on a appelée *indices implicites*, permettant d'entraîner un algorithme de renforcement utilisant une mémoire de reprise avec un réseau de neurones et des tailles d'images beaucoup plus grands que la majorité des travaux précédents de renforcement.

— Une étude extensive des paramètres et des études d'ablations sur les *indices implicites* et la fonction de récompense.

— Démonstration de la capacité de notre méthode en remportant le *CARLA Autonomous Driving Challenge* en 2019 et en 2020 sur la catégorie "Cameras Only", c'est à dire en n'utilisant que des informations caméras pour conduire.

5.2 Travaux connexes

Nos contributions s'intéressent principalement à comment apprendre une représentation suffisamment sémantique et compacte pour pouvoir être utilisée comme état d'un agent entraîné par DRL pour la conduite urbaine. C'est pourquoi nous avons divisé les travaux connexes en deux parties. Dans la première, nous nous intéresserons à l'apprentissage de représentation pour le renforcement et nous verrons que certains articles récents sont passés par des *indices* en particulier pour la voiture autonome. Dans un deuxième temps, nous détaillerons les quelques articles récents appliquant du renforcement à la conduite urbaine, l'ensemble de ces articles s'appliquant pour le moment dans le simulateur CARLA car comme indiqué en partie 2.5.1, CARLA est le simulateur open-source d'environnement urbain le plus abouti.

5.2.1 Apprentissage de représentation pour le renforcement

L'agent UNREAL [Jaderberg et al., 2016] est un des premiers articles étudiant l'impact des tâches auxiliaires pour l'apprentissage par renforcement profond. Dans cet article, les chercheurs ont montré que rajouter des *coût auxiliaires* comme prédire la prochaine récompense pouvait améliorer l'efficacité en donnée ainsi que les performances finales d'un agent entraîné avec l'algorithme A3C [Mnih et al., 2016] (voir section 2.4.2). Il est intéressant de noter que pour ce travail, Jaderberg et al. [2016] se sont intéressés à des tâches auxiliaires très génériques, c'est à dire qu'elles sont utilisables indépendemment de l'application sur laquelle on veut entraîner l'agent. Ces tâches auxiliaires sont : la prédiction de la récompense future, la prédiction de la fonction de valeur ainsi que l'effet de l'action prise sur l'état suivant. Elles sont illustrées sur la Figure 5.1.

Ce résultat est donc très encouragent, même des tâches auxiliaires très génériques permettent d'améliorer l'efficacité et les performances finales sur différentes applications, le benchmark Atari ainsi que des tâches de navigations dans un labyrinthe.

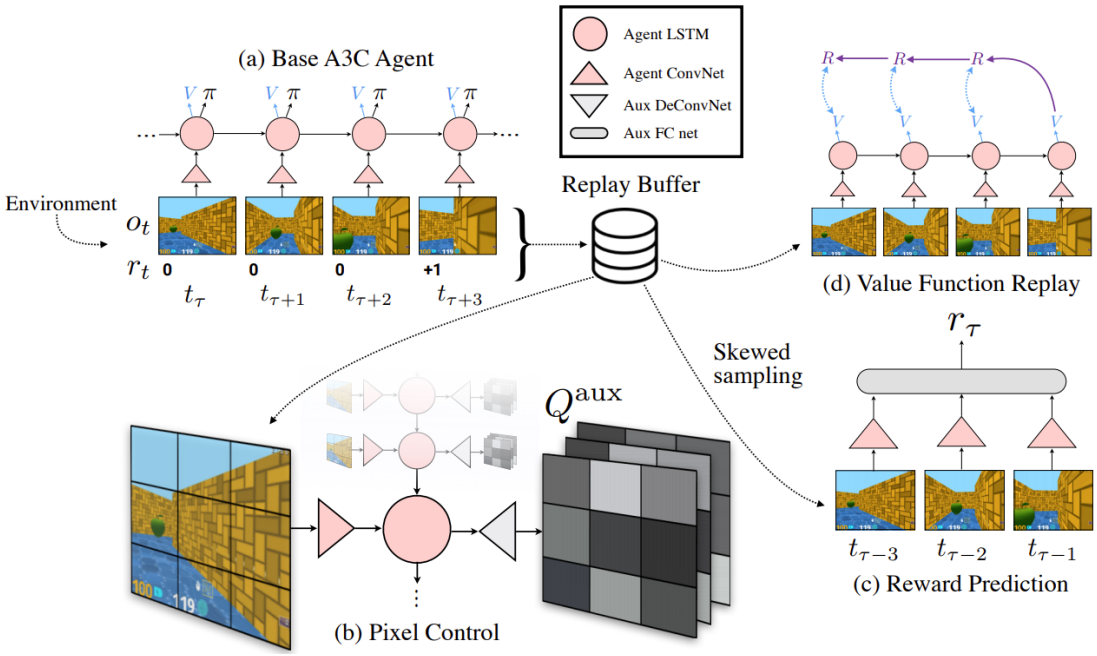


FIGURE 5.1 – Schéma représentant les différentes tâches auxiliaire utilisées par Jaderberg et al. [2016] dans l'article UNREAL. Image provenant de Jaderberg et al. [2016].

Un travail concurrent au nôtre et qui implique une méthode très proche est l'article SplitNet [Gordon et al., 2019]. Dans ce travail, les chercheurs découplent eux aussi l'apprentissage de bout-en-bout en deux phases : tout d'abord l'apprentissage de caractéristiques utiles pour la perception, puis l'apprentissage d'une politique comportementale. Gordon et al. [2019] vont comme nous d'abord entraîner l'encodeur par apprentissage supervisé puis utilisent les features issues de cet encodeur comme état pour l'agent appris par renforcement. Ils ont appliqué leur principe pour le transfert d'apprentissage, à la fois le transfert visuel d'un simulateur à un autre où la tâche est commune ainsi que le transfert de tâche, i.e. passer d'une tâche d'exploration à une tâche de navigation. Une illustration de leur travail est fournie en Figure 5.2.

Il existe de nombreux autres travaux dans ce domaine, *l'apprentissage de représentation* [Yarats et al., 2019, Higgins et al., 2017] cherchant à améliorer l'efficacité des algorithmes de renforcement. Nous allons maintenant nous intéresser aux travaux s'appliquant spécifiquement à la voiture autonome.

Un autre travail concurrent du nôtre et très proche de notre méthode est l'article de Pan et al. [2019]. Dans cet article, Pan et al. [2019] vont eux-aussi entraîner un réseau de neurones à estimer des informations haut niveau importantes pour la conduite, comme

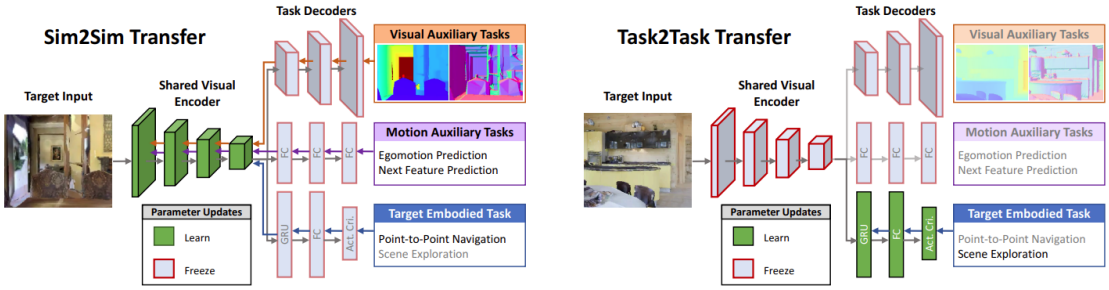


FIGURE 5.2 – Schéma représentant le principe de SplitNet [Gordon et al., 2019]. A gauche, le but est de transférer un apprentissage d'un simulateur à l'autre en fixant les poids de la politique et en ré-apprenant seulement les poids de l'encodeur. A droite, pour le transfert d'une tâche à l'autre, Gordon et al. [2019] fixent les poids de l'encodeur et n'entraînent que la politique. C'est surtout cette deuxième partie qui ressemble à notre technique d'*indices implicites*. Image provenant de Gordon et al. [2019].

la carte de segmentation sémantique ou la probabilité de collision (voir Figure 5.3 pour la liste de ces informations haut niveau). Ils vont ensuite utiliser ces informations dans le cadre d'un apprentissage par renforcement fondé sur un modèle de l'environnement. L'idée est donc de faire de la planification en prévoyant sur plusieurs pas de temps la probabilité de sortir de sa voie, de collision etc... afin de choisir l'action menant à la plus grande récompense. Cependant, leur approche diverge de la nôtre car premièrement ils appliquent de l'apprentissage par renforcement fondé sur un modèle alors que nous avons fait de l'apprentissage *sans modèle*. De plus leur cas d'application est plus simple et ne gère ni les intersections ni les feux tricolores.

5.2.1.1 Apprentissage d'*affordances* pour la voiture autonome

Le travail de Chen et al. [2015] est le premier à introduire le terme d'*affordances* pour la voiture autonome : un réseau de neurones est entraîné à estimer des informations haut niveau comme la distance par rapport au centre de la voie ou bien la distance du véhicule situé devant. Le terme d'*affordance* a une définition très précise dans le domaine de la psychologie cognitive [Luyat and Regia-Corte, 2009] ou de l'interaction homme-machine qui ne correspond pas totalement à ce qu'utilisent Chen et al. [2015] dans leur travail. Nous avons donc choisi de traduire ce terme par *indice* tout au long de ce manuscrit mais nous avons gardé le terme d'affordance dans notre article en anglais pour s'aligner avec l'état de l'art. Chen et al. [2015] ont ensuite utilisé ces indices en entrée d'un contrôleur fondé sur des règles et ont atteint de bonnes performances dans le simulateur de course TORCS [Wymann et al., 2000]. Sauer et al. [2018] ont amélioré ce principe dans leur travail Conditionnal Affordance Learning (CAL) [Sauer et al., 2018] pour gérer des situations plus complexes de conduite urbaine. Pour cela, ils ont rajouté d'autres indices à apprendre pour le réseau de neurones comme la vitesse maximale autorisée ou bien la présence et l'état d'un feu tricolore, cf Figure 5.4. Comme Chen et al. [2015], ils

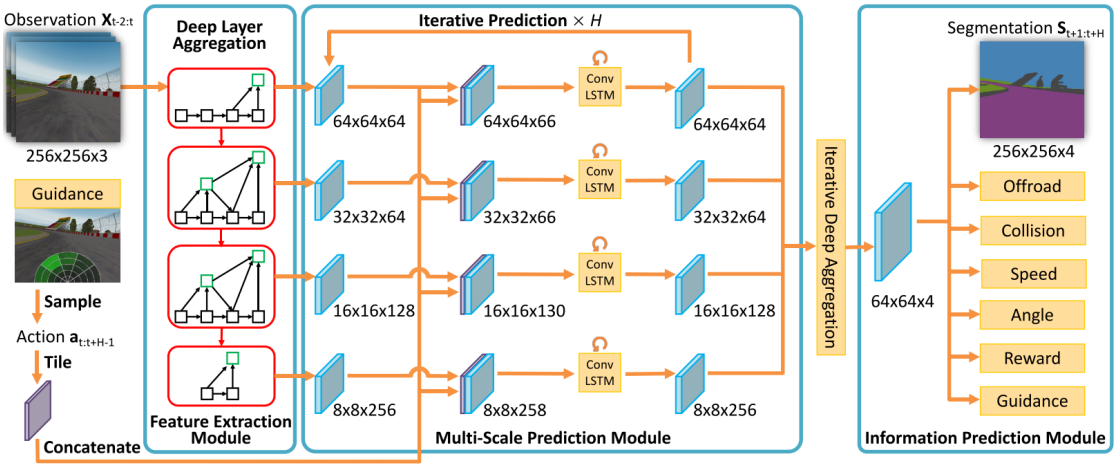


FIGURE 5.3 – Apprentissage par renforcement fondé sur un modèle de l'environnement appliqué à la conduite autonome. Tout d'abord Pan et al. [2019] entraînent un réseau à estimer des informations haut niveau comme la probabilité de collision ou de sortir de sa voie, la carte de segmentation sémantique etc... L'idée est ensuite d'utiliser ces informations pour faire de la planification sur plusieurs pas de temps. Finalement, en utilisant une fonction de coût faite à la main les chercheurs appliquent l'action qui mène au coût le plus faible (en imaginant les futures trajectoires). Image provenant de Pan et al. [2019].

ont ensuite utilisé ces estimations en entrée d'un contrôleur classique et ont démontré leur performance sur le benchmark CARLA (c.f section suivante 5.3) pour la conduite urbaine.

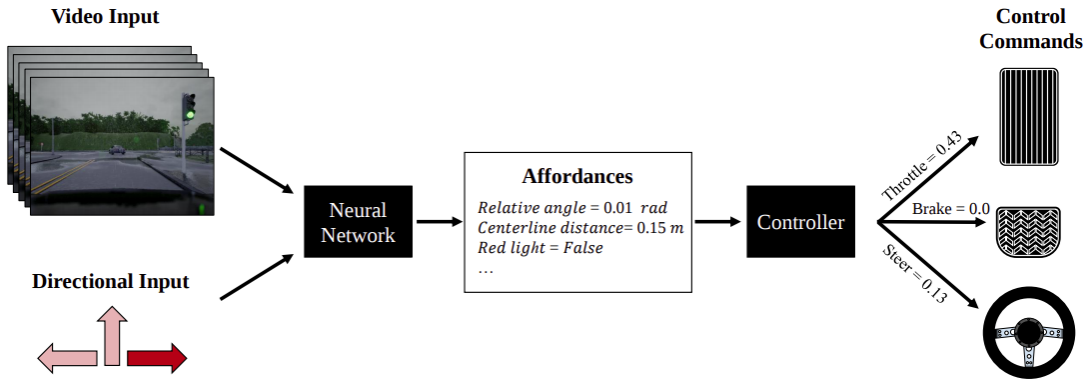


FIGURE 5.4 – Conditionnal Affordances Learning, Sauer et al. [2018] estiment différents *indices* comme la distance au centre de la voie ou bien l'état du feu tricolore. Ces informations sont ensuite utilisées par un contrôleur classique pour commander le véhicule. Image provenant de Sauer et al. [2018].

Nous nous sommes grandement inspirés de ces idées de *indices* dans notre approche. En effet, notre première phase est comparable à la phase d'apprentissage de CAL [Sauer et al., 2018] par exemple. Cependant, notre approche va totalement se passer du contrôleur classique pour le remplacer par un apprentissage par renforcement sans modèle, ce qui n'avait jamais été fait auparavant pour la conduite en environnement urbain.

5.2.2 Application du DRL à la conduite urbaine de bout-en-bout

Dans cette section, nous détaillerons brièvement les articles récents de Jianyu Chen [Chen et al., 2019, 2020b] appliquant de l'apprentissage par renforcement sans modèle pour la conduite urbaine, le deuxième étant une extension du premier travail. A notre connaissance, ce sont les seuls autres articles appliquant du renforcement sans modèle pour la conduite urbaine. Ils reposent eux aussi sur le simulateur CARLA.

Le premier article que nous détaillerons [Chen et al., 2019] n'est à proprement parler pas de bout-en-bout : en effet les chercheurs utilisent en entrée de leur réseau une image vue du dessus avec des informations haut niveau. Ils donnent ainsi les positions passées des véhicules environnants, l'historique des positions passées de l'agent ainsi qu'une information sur la position des voies et la voie à suivre. Ces informations sont illustrées Figure 5.5.

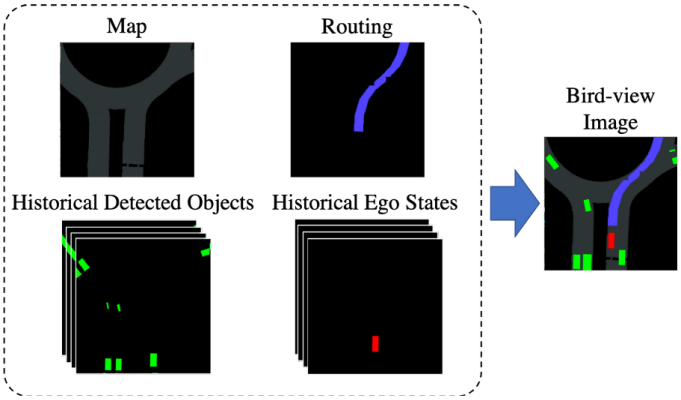


FIGURE 5.5 – Représentation utilisée en entrée d'un apprentissage par renforcement sans modèle pour la conduite en environnement urbain [Chen et al., 2019]. Ce n'est donc pas un apprentissage de bout-en-bout car la majeure partie de la perception provient d'informations privilégiées issues du simulateur CARLA. Image provenant de Chen et al. [2019].

Même si l'information est beaucoup plus compacte que les données brutes des capteurs, elles restent encore de grande dimension (la taille de l'image vue du dessus). C'est pour cela que Chen et al. [2019] ont utilisé un VAE (pour Variational AutoEncoder [Kingma and Welling, 2013]) afin de réduire encore la dimension de l'entrée qui sera utilisée comme état pour l'agent appris par renforcement. Un auto-encodeur, par exemple un VAE, est composé d'un encodeur et d'un décodeur. Le but du décodeur est de re-

construire l'entrée en utilisant uniquement les features issues de l'encodeur qui sont de bien moins grande dimension que l'entrée. L'intuition est donc de condenser l'information dans un espace de dimension plus faible mais qui reste suffisant pour contenir toute l'information initiale car le décodeur du VAE doit être capable de reconstruire l'entrée initiale uniquement à partir des caractéristiques du *goulet d'étranglement*. Chen et al. [2019] vont ensuite tester et comparer différents algorithmes de renforcement de l'état de l'art en utilisant les features issues de l'encodeur du VAE comme état. Cependant, dans ce premier travail, ils ne testent leur algorithme que sur un cas très spécifique d'insertion et de sortie d'un même rond-point. On est donc très loin du cas d'application sur lequel nous travaillons, en particulier ils ne prennent en compte ni les feux tricolores, ni les piétons.

Plus récemment, juste après notre publication, le même chercheur a publié une extension de ce premier travail adapté à un cas d'application plus large [Chen et al., 2020b]. De plus, dans cette extension, l'apprentissage se fait bien de bout-en-bout dans le sens où ils apprennent les informations haut niveau utilisées précédemment (illustrées Figure 5.5) et utilisent l'estimation de ces informations haut niveau comme tâche auxiliaire pour leur agent entraîné par renforcement. L'architecture générale de leur apprentissage bout-en-bout est représenté en Figure 5.6. Les chercheurs comparent aussi différents algorithmes de renforcement sans modèle, en particulier SAC [Haarnoja et al., 2018] qui est actuellement l'algorithme état de l'art sur le benchmark de contrôle continu MuJoCo [Todorov et al., 2012].

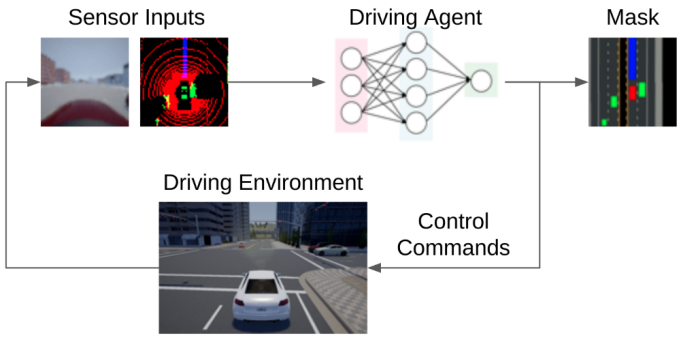


FIGURE 5.6 – Architecture de l'apprentissage bout-en-bout de l'article de Chen et al. [2020b]. Le réseau de neurones doit calculer un contrôle pour le véhicule ainsi qu'effectuer une tâche auxiliaire de perception. Cette tâche auxiliaire de perception implique la détection des objets mobiles, des voies de circulation ainsi que le *routing*, i.e. quelle voie doit suivre l'agent. Image provenant de Chen et al. [2020b].

Cependant, il est important de noter qu'ils ne prennent toujours pas en compte les feux tricolores et qu'ils n'ont fait aucune comparaison à d'autres références, en particulier sur le benchmark CARLA que nous décrirons dans la section suivante. De plus, il est difficile de comprendre dans leur article comment est choisie la commande (qu'ils appellent *routing*) indiquant à l'agent quelle voie il doit suivre dans une intersection : aller à droite, tout droit ou à gauche.

5.3 Le benchmark CARLA et le challenge CARLA

5.3.1 Le benchmark CARLA

L'article original de CARLA [Dosovitskiy et al., 2017] présentant les principales caractéristiques et intérêts du simulateur a aussi introduit un ensemble de tâche pour évaluer des algorithmes de conduite autonome : le *benchmark CARLA*. Les tâches de ce benchmark sont des tâches de navigation où l'agent doit parcourir un trajet défini par un point de départ et d'arrivée. Pour cela l'agent a accès à différents capteurs, caméras et LIDARs, ainsi qu'à des commandes haut niveau indiquant quelle voie prendre aux intersections (droite, gauche ou tout droit). Ces tâches sont séparées en quatre catégories de difficulté croissante : *tout droit*, *un seul virage*, *navigation libre* et *navigation libre avec objets dynamiques*. La seule différence entre les trois premières catégories est le nombre de virages à franchir entre le point de départ et le but final, comme illustré Figure 5.7. *Tout droit* indique qu'il n'y a aucun virage même si il peut malgré tout y avoir des intersections où l'agent doit aller tout droit.

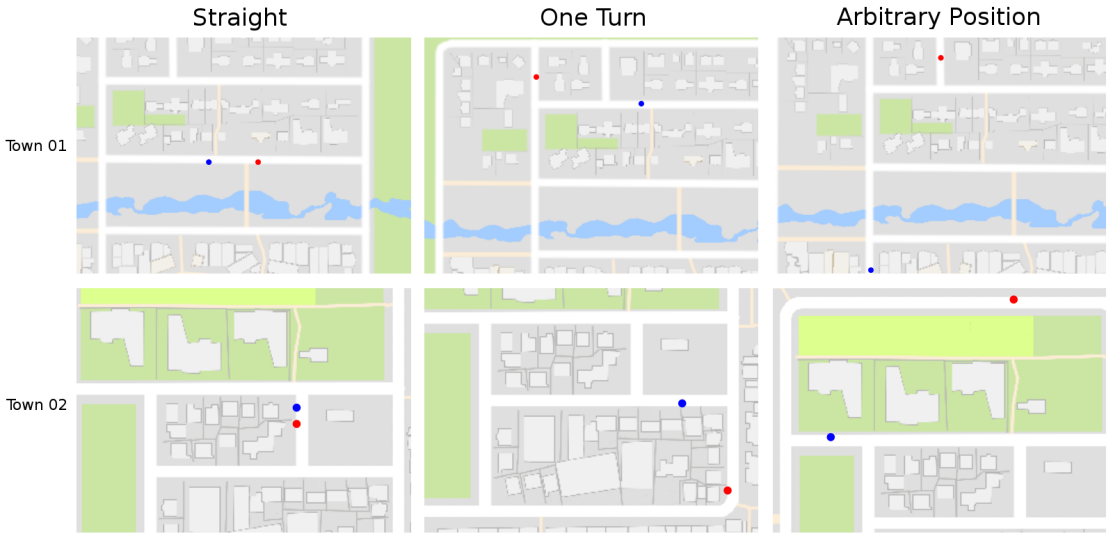


FIGURE 5.7 – Exemple de tâche de navigation du benchmark CARLA [Dosovitskiy et al., 2017], la difficulté est croissante de gauche à droite : *Tout droit* à gauche, *Un seul virage* au milieu et *navigation libre* à droite. Le point bleu correspond au point de départ de l'agent, le point rouge est l'endroit où doit arriver l'agent pour réussir la tâche. Image provenant de Dosovitskiy et al. [2017].

Les deux dernières catégories sont les plus ardues et les points de départ et d'arrivée sont placés aléatoirement sur la carte. La seule de ces catégories comportant des objets dynamiques, i.e. des véhicules et des piétons est la dernière catégorie : *navigation libre avec objets dynamiques*. Étant l'un des seuls voire même le seul benchmark en libre accès pour évaluer des algorithmes de conduite, ce benchmark a été utilisé comme référence

pour évaluer et comparer les performances entre différents articles comme Sauer et al. [2018], Chen et al. [2020a], Toromanoff et al. [2020]. Cependant, ce benchmark comporte de nombreux points faibles. En effet, l'évaluation finale ne dépend que de si l'agent a atteint le point d'arrivée : une tâche est réussie si et seulement si l'agent arrive au point d'arrivée. Cela ne dépend aucunement de la manière pour arriver au point d'arrivée. Par exemple les collisions, les feux tricolores, les dépassements de voies ne sont pas pris en compte, ce qui semble très loin de l'objectif final voulu de la voiture autonome. De plus, l'environnement de test est totalement accessible, on pourrait donc en théorie *tricher* et apprendre directement sur l'environnement de test pour améliorer ses performances. C'est pour cela que les développeurs du simulateur CARLA ont introduit une amélioration de ce benchmark : le challenge CARLA.

5.3.2 Le challenge CARLA

Le challenge CARLA [2] est à l'origine de l'ensemble des travaux de ce chapitre. Nous avons en effet d'abord participé à ce challenge et après avoir obtenu d'excellents résultats, nous avons publié nos travaux à la conférence CVPR. Tout comme le benchmark CARLA, le challenge CARLA consiste en des tâches de navigation avec des commandes haut niveau pour indiquer quelle voie prendre aux intersections. Cependant, il y a de nombreux points qui le rendent bien plus complexe que le benchmark initial.

Premièrement, l'évaluation est beaucoup plus fine et prend en compte l'ensemble des infractions commises par l'agent que ce soit les collisions, les sorties de voies ou bien le non-respect des feux tricolores. De plus, l'environnement de test a été gardé secret et cela induit une capacité de généralisation bien plus grande. Une autre difficulté comparée au benchmark initial est la présence de routes comportant plusieurs voies et l'ajout de commande demandant à l'agent de changer de voie : il y a donc deux commandes supplémentaires correspondant au changement de voie droite et gauche. Finalement, l'agent doit gérer des feux tricolores européens et américains en même temps. En effet, les feux sont placés de l'autre coté de l'intersection aux État-unis alors qu'en Europe ils sont placés directement devant l'intersection, comme illustré Figure 5.8. Cela est en fait loin d'être anodin, particulièrement parce que les feux à l'américaine sont plus loin et donc plus difficiles à détecter. Nous verrons dans la prochaine partie que c'est surtout à cause des feux à l'américaine que l'on a dû utiliser une image d'entrée beaucoup plus grande que celle habituellement utilisée dans les travaux d'apprentissage par renforcement profond.

Le challenge CARLA comporte quatre catégories qui se différencient uniquement par les entrées accessibles à l'agent. Ces quatre catégories sont détaillées sur la Figure 5.9.

Nous nous sommes principalement intéressés à la catégorie la plus complexe ("Cameras Only") sur laquelle l'agent n'a accès qu'à des images caméras pour percevoir le monde. Nous avons finalement remporté la première place sur cette catégorie ainsi que la deuxième place sur la catégorie *caméras et LIDARs* malgré le fait que nous n'utilisions pas de LIDAR !

2. https://carlachallenge.org/

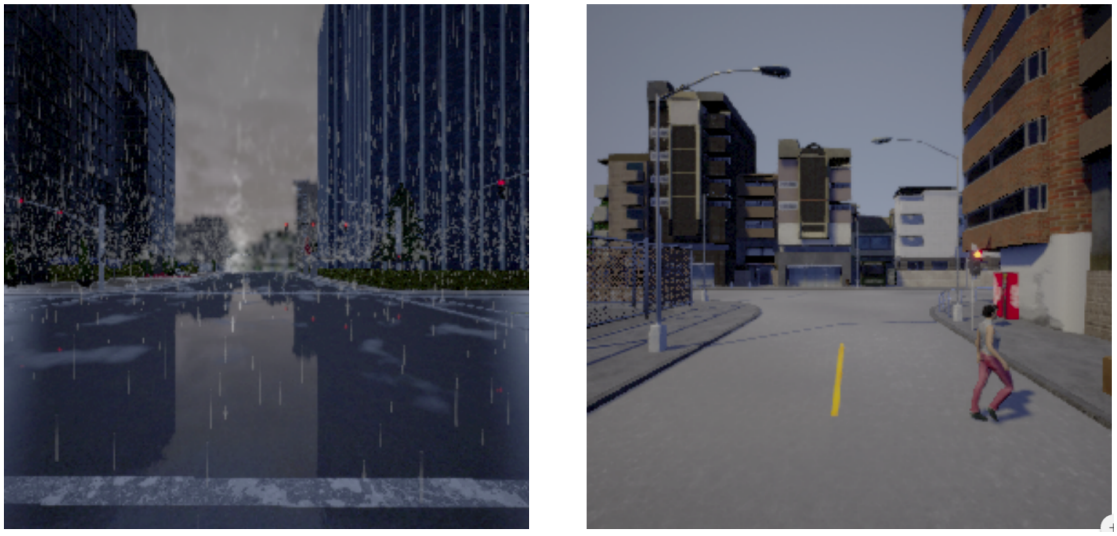


FIGURE 5.8 – Exemple de situation avec des feux tricolores dans CARLA. A gauche feu à l'américaine et à droite feu à l'européenne. Les feux sont beaucoup plus difficiles à détecter sur les intersections à l'américaine car ils sont plus loin de la où l'agent doit s'arrêter.

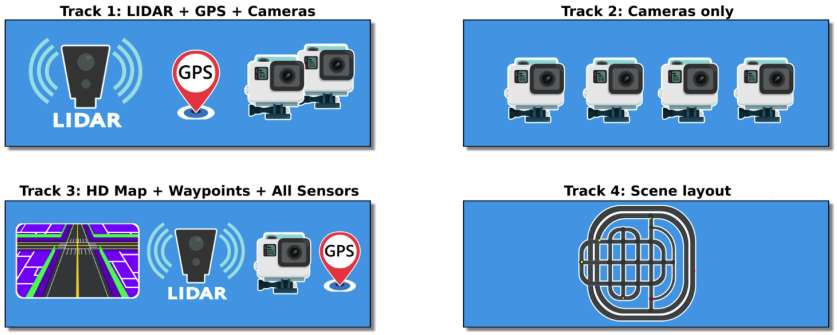


FIGURE 5.9 – Les quatre *catégories* possibles du challenge CARLA. La seule différence entre chaque catégorie est les entrées accessibles à l'agent. Du plus difficile comportant uniquement des caméras (catégorie 2) au plus simple où toute la perception, e.g. position/vitesse des véhicules, état de tous les feux de signalisation, est donnée sur une vue du dessus (catégorie 4).

5.4 Méthodes

Dans cette partie nous allons décrire notre approche générale, en particulier sur notre structure d'apprentissage par renforcement, sur la forme de notre fonction de récompense ainsi que l'architecture de notre réseau. Nous verrons dans la partie suivante quelles ont été les adaptations nécessaires pour rendre notre approche possible en pratique dans le cadre d'une tâche complexe de conduite en environnement urbain.

5.4.1 Apprentissage par renforcement fondé sur la fonction de valeur : Rainbow-IQN Ape-X

Comme indiqué dans le Chapitre 2, les algorithmes fondés sur la fonction de valeur semblent les plus prometteurs pour l'application à la voiture autonome en particulier pour leur caractère *off-policy* leur permettant d'être plus *efficace en données*. Nous avons présenté notre algorithme Rainbow-IQN qui a dépassé l'état de l'art actuel sur le benchmark Atari dans le chapitre précédent, Chapitre 4. Nous avons aussi introduit notre version distribuée de cet algorithme, Rainbow-IQN Ape-X, mais qui n'a pas été réellement utilisée pour l'évaluation sur le benchmark Atari. Nous avions implémenté cette version distribuée dans l'optique de l'appliquer à la voiture autonome où la génération de donnée est beaucoup plus coûteuse que l'émulateur Atari.

Cette version distribuée a ainsi été absolument nécessaire pour tous les travaux de ce chapitre : le simulateur CARLA est assez lent et ne peut pas générer suffisamment de données si une seule instance est utilisée. De plus, cela nous a permis d'entraîner nos agents sur plusieurs cartes de CARLA en même temps, ce qui génère plus de variabilité dans les données d'entraînements, améliore l'exploration et surtout nous a donné une manière simple de gérer les feux américains et européens. En effet, durant l'entraînement certaines villes étaient américaines et d'autre européennes. Cela permet à l'agent d'avoir les 2 types de feux tricolores à gérer en même temps.

5.4.2 Choix de la fonction de récompense

Un autre aspect très important de notre structure d'apprentissage par renforcement est le choix de la fonction de récompense. Pour définir notre fonction de récompense, nous nous sommes principalement reposés sur l'API de CARLA permettant d'accéder au tracé des différentes voies. Cette API permet d'obtenir les positions et les rotations de chaque voie dans la ville simulée. Cela est indispensable pour décider du chemin que doit suivre l'agent. De plus, cette API permet de savoir quelles sont les différentes possibilités à chaque intersection. Ainsi au début d'un épisode, l'agent est initialisé à un endroit aléatoire de la ville, puis la trajectoire optimale que devrait suivre l'agent peut être calculée en utilisant l'API de CARLA. Lorsque l'agent arrive à une intersection, nous choisissons aléatoirement une manoeuvre possible (à gauche, tout droit ou à droite) et la commande haut niveau associée est donnée à l'agent. Notre récompense finale se compose de 3 termes : la *vitesse désirée*, la *position désirée* ainsi que la *rotation désirée*.

La récompense associée à la *vitesse désirée* est maximale (et égale à 1) lorsque l'agent conduit à la vitesse désirée, et descend linéairement jusqu'à 0 si la vitesse de l'agent est plus faible ou plus grande. L'intérêt majeur de cette récompense est que l'on va adapter la *vitesse désirée* à la situation comme illustré en Figure 5.10. Quand l'agent arrive à un feu rouge, la *vitesse désirée* descend linéairement jusqu'à 0 le plus proche l'agent est du feu rouge, et elle repasse à la vitesse maximale autorisée lorsque le feu passe au vert. Le même principe est appliqué lorsque l'agent arrive derrière un obstacle, un piéton, un vélo ou une voiture. La *vitesse désirée* est réglée à une *vitesse maximale* constante (qu'on a mise à 40km/h) dans toutes les autres situations.

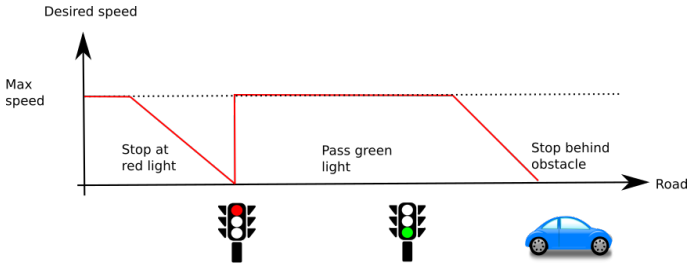


FIGURE 5.10 – Vitesse désirée en fonction de l'environnement. La vitesse désirée est adaptée à la situation, diminuant lorsque l'agent se rapproche d'un feu rouge, remise à la vitesse maximale lorsque le feu passe au vert et de même diminue lorsque l'agent se rapproche d'un obstacle. La récompense associée est maximale égale à 1 lorsque l'agent conduit à la vitesse désirée.

Le deuxième terme de la récompense, la *position désirée*, est inversement proportionnel à la distance au centre de la voie (calculée à l'aide de l'API Carla). Cette récompense est maximale égale à 0 lorsque l'agent est exactement au milieu de sa voie et diminue jusqu'à -1 quand l'agent atteint une distance maximale de sa voie D_{max}. Lorsque l'agent est trop loin de sa voie, l'épisode se termine. Pour toute nos expériences, D_{max} était égal à 2 mètres : cela correspond environ à la distance du centre de la voie à la bordure. Les autres conditions pour terminer un épisode sont les collisions avec tout objet, griller un feu rouge ou bien rester bloqué sans raison, i.e. quand il n'y a ni obstacle ni feu rouge devant l'agent. Pour toutes ces conditions de fin d'épisode, l'agent reçoit une récompense égale à -1.

Initialement notre récompense ne comportait que ces deux premiers termes mais nous avons observé que l'agent oscillait autour du centre de sa voie. En effet, de petites oscillations amènent des récompenses quasiment identiques à aller tout droit. C'est pourquoi nous avons rajouté un troisième terme, la *rotation désirée*. Cette récompense est inversement proportionnelle à la différence d'angle entre l'agent et l'orientation du point le plus proche de sa voie. Le calcul des 2 récompenses, *position désirée* et *rotation désirée*, est illustré en Figure 5.11. Des expériences d'ablations sur le choix de la fonction de récompense sont présentées en section 5.6.

5.4.3 Architecture de notre réseau de neurones

Comme dit précédemment, la majorité des réseaux de neurones utilisés pour l'apprentissage par renforcement sont des réseaux particulièrement petits [Mnih et al., 2015, 2016] comparés aux réseaux utilisés communément en apprentissage supervisé [Simonyan and Zisserman, 2014, He et al., 2016]. Un des réseaux les plus lourds utilisé pour du RL est la *large architecture* de l'article IMPALA [Espeholt et al., 2018] qui consiste en 15 couches de convolution pour environ 1.6 million de paramètres : à titre de comparaison, notre architecture comporte 18 couches de convolution pour environ 30 millions de paramètres. Il est important de noter que le *large network* de IMPALA a nécessité plus

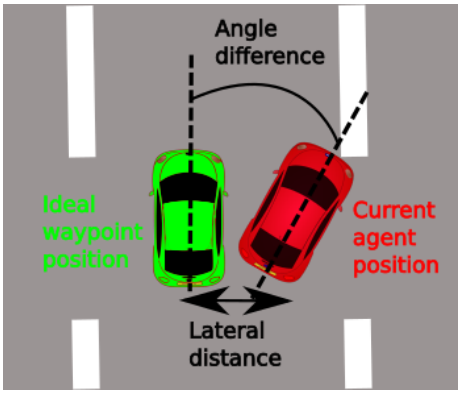
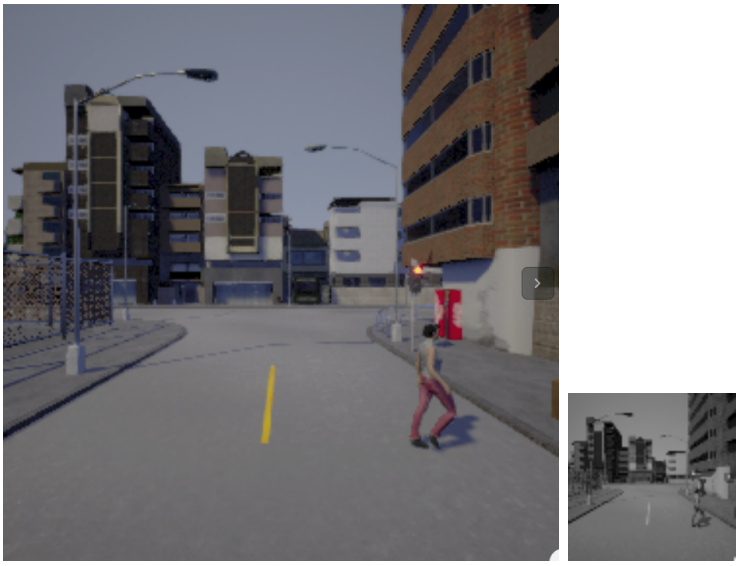


FIGURE 5.11 – Illustration des récompenses *position désirée* et *rotation désirée*. L'idée est de se comparer à la position et à la rotation optimale du point de la voie le plus proche. L'information de position/rotation du point de passage nous est fournie par l'API de CARLA.

d'un milliard d'images pour converger quand nous avons utilisé seulement 20 millions d'images.



(a) Image RGB 288x288x3 (b) Image 84x84

FIGURE 5.12 – A gauche notre taille d'image finale, on distingue sans problème l'état du feu à l'européenne. A droite, la taille d'image utilisée par la majeure partie des travaux précédents en DRL [Mnih et al., 2015, Hessel et al., 2018]. Même les feux à l'européenne sont très durs à distinguer, les feux à l'américaine sont donc totalement indétectables sur une aussi petite image.

L'architecture de réseau la plus commune dans les travaux de renforcement est celle

introduite dans l'article original de DQN [Mnih et al., 2015], qui prend en entrée une image en niveau de gris de très petite taille (84x84). Notre première observation est que l'état des feux tricolores, particulièrement les feux à l'américaine, ne peuvent pas être vus sur de si petites images (cf Figure 5.12). Pour cette raison, nous avons choisi une taille d'entrée beaucoup plus importante (environ 40 fois plus grande) : 4x288x288x3. Nous concaténons 4 images RGB consécutives car c'est une méthode simple et standard [Mnih et al., 2015, 2016] pour ajouter de l'information temporelle à l'entrée du réseau. Nous avons choisi cette taille d'image car c'était la plus petite sur laquelle nous avions des performances encore bonnes pour la détection de l'état du feu de signalisation (en utilisant un apprentissage supervisé standard).

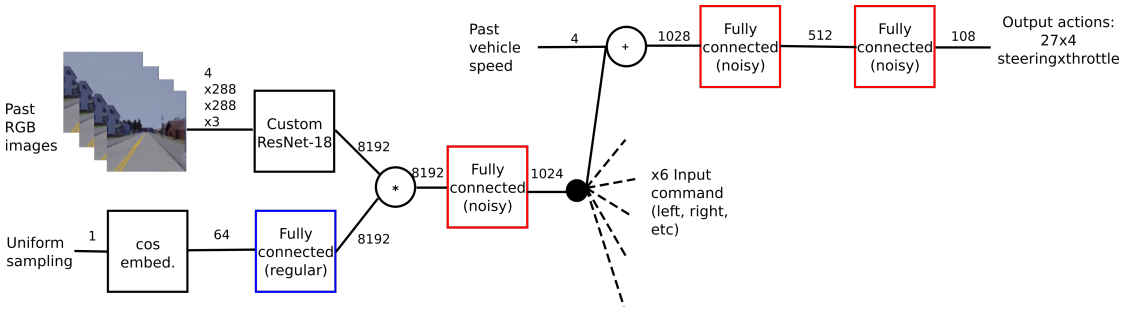


FIGURE 5.13 – Architecture de notre réseau de neurones. Un encodeur Resnet-18 [He et al., 2016] est utilisé dans un réseau conditionnel [Codevilla et al., 2018] permettant de prendre en compte les 6 commandes de navigation. L'entraînement par renforcement utilise l'algorithme Rainbow-IQN (d'où proviennent la partie IQN [Dabney et al., 2018] et les *couches FC (Fully Connected) bruitées* [Fortunato et al., 2017].

Concernant l'architecture du réseau elle-même, nous avons choisi d'utiliser un Resnet-18 [He et al., 2016] car c'est un réseau relativement petit (comparé à ceux utilisés communément pour l'apprentissage supervisé) avec un temps d'inférence faible. L'apprentissage par renforcement nécessite en effet une quantité très importante de données, chaque pas d'entraînement doit donc être le plus rapide possible afin de réduire le temps global de l'apprentissage. Cependant, même si Resnet-18 est parmi les réseaux les plus légers utilisés pour l'apprentissage supervisé, il contient environ 140 fois plus de poids que le réseau utilisé pour DQN. De plus, ce réseau incorpore la plupart des avancées récentes en apprentissage supervisé comme les connections résiduelles ou bien la *batch normalisation* [Ioffe and Szegedy, 2015]. Finalement, nous utilisons un réseau de neurones *conditionnel* comme dans l'article de Codevilla et al. [2018] pour gérer les 6 manoeuvres possibles : rester sur sa voie, intersection à gauche/tout droit/droite et changer de voie gauche/droite. L'architecture globale de notre réseau de neurones est détaillée sur la Figure 5.13

5.5 Défis et solutions pour appliquer du RL à la conduite urbaine

Dans cette section, nous présentons nos suggestions pour résoudre les problèmes survenant lorsque l'on veut entraîner un réseau avec un grand nombre de paramètres par renforcement. Nous verrons aussi comment gérer les actions discrètes dans le cadre du contrôle d'un véhicule.

5.5.1 Entraîner par renforcement un agent ayant des entrées de haute dimension : *indices implicites*

Comment entraîner un plus grand réseau par renforcement avec des images en haute résolution ? Il y a deux problèmes majeurs qui surviennent lorsque l'on utilise un réseau et des tailles d'images importantes. Le premier problème est qu'un tel réseau est beaucoup plus long et dur à faire converger. Le deuxième problème est la mémoire de reprise. L'un des avantages principaux des algorithmes fondés sur la fonction de valeur comparé à ceux fondés sur la politique est qu'ils sont *off-policy*, ce qui permet de réutiliser plusieurs fois une même expérience pour la mise à jour des poids du réseau. Cependant, comme les images sont maintenant environ 40 fois plus grandes, il peut y avoir un problème de stockage pour avoir le même nombre d'expériences dans la mémoire de reprise. Par exemple, la mémoire de reprise de DQN comporte un million d'images, ce qui correspond à 6 GB pour des images 84x84, mais pour garder le même nombre d'images, il faudrait allouer plus de 200 GB de mémoire vive avec des images 40 fois plus grandes ce qui est peu pratique.

Notre idée principale est de pré-entraîner la partie encodeur du réseau (qui comporte l'ensemble des couches convolutionnelles) à estimer des informations haut niveau, puis de fixer ses poids durant l'entraînement par renforcement de l'agent. Cela permet de compresser la dimension des images d'entrée tout en s'assurant que les informations pertinentes sont contenues dans les features car le décodeur doit être capable d'estimer les indices de conduite. L'intuition est que le signal d'apprentissage du renforcement est trop faible pour entraîner le réseau dans son ensemble mais peut être utilisé pour entraîner seulement la dernière partie du réseau comportant les couches entièrement connectées. De plus, cela résout aussi le problème avec la mémoire de reprise car on peut maintenant enregistrer seulement les features issues de l'encodeur et non plus les images brutes. Nous avons appelé ce procédé *indices implicites* car l'agent n'a pas accès aux estimations explicites des indices effectuées par le décodeur mais a seulement accès aux indices implicites : les features calculées par l'encodeur permettant à notre décodeur initial d'estimer les indices explicites.

Quelles indices/informations sémantiques haut niveau estimer ? L'idée la plus simple pour pré-entraîner notre encodeur serait d'utiliser un auto-encodeur [Kingma and Welling, 2013], i.e. essayer de compresser nos images d'entrée en reconstruisant les images d'entrée à partir d'un espace de features de plus petite dimension. Cela a par exemple

été utilisé dans l'article de Kendall et al. [2019] et a permis d'accélérer l'entraînement sur une voiture réelle pour faire du maintien de voie. Nous pensons que cette approche ne pourrait pas fonctionner dans notre cas d'application complexe particulièrement par rapport à la détection des feux tricolores. En effet, la couleur du feu ne représente qu'une partie infime de l'image (rouge, orange ou vert) alors que ces quelques pixels sont les plus importants pour le comportement à suivre.

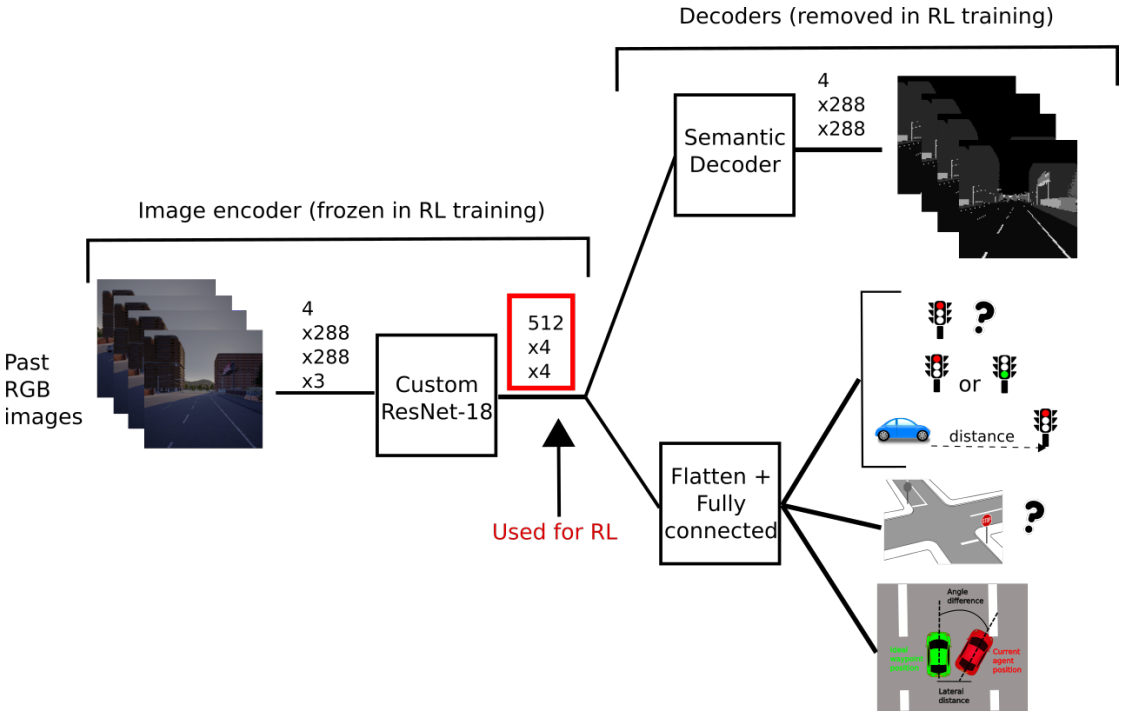


FIGURE 5.14 – Pré-entraînement de l'encodeur par apprentissage supervisé avec toutes les fonctions de coût que l'on a utilisées. L'idée est ensuite de fixer les poids de l'encodeur et d'utiliser les features intermédiaires comme état pour l'agent appris par renforcement au lieu des images brutes.

Nous avons choisi d'utiliser des informations sémantiques haut niveau accessibles dans CARLA pour s'assurer qu'il y ait les informations nécessaires dans les features qu'on utilise en tant qu'état d'entrée pour l'agent entrainé par RL. Nous avons utilisé deux fonctions de coût principales pour notre phase d'apprentissage supervisé : la détection de l'état du feu de signalisation, ainsi que l'estimation d'une segmentation sémantique de l'image. En effet, la majorité des informations pertinentes sont comprises dans la carte de segmentation sémantique, sauf l'état du feu. A noter qu'on aurait pu aussi utiliser différentes classes selon l'état du feu dans la carte de segmentation mais cela nous a semblé plus difficile à apprendre pour le réseau. Nous avons utilisé six classes pour notre masque de segmentation : les objets mobiles, les feux tricolores, les marquages au sol, la route, les trottoirs et finalement l'arrière plan. Nous avons aussi rajouté d'autre indices à estimer pour aider l'entraînement supervisé comme la distance au feu tricolore, si

l'agent est sur une intersection ou non, la distance au centre de la voie ainsi que la rotation de l'agent par rapport à la route. A noter que l'enregistrement de la base de données nécessaire à cet apprentissage supervisé s'est fait à partir de la conduite d'un expert dans CARLA. Cet expert est un pilote automatique du simulateur CARLA qui a accès à toutes les informations nécessaires. Notre entraînement supervisé avec toutes les informations haut niveau et les indices à estimer est représenté sur la Figure 5.14.

Augmentation de point de vue lors de l'enregistrement de la base de données pour l'apprentissage supervisé Comme indiqué précédemment, les données ont été collectées en conduisant avec un pilote automatique pré-existant dans le simulateur CARLA. Cependant, cet expert reste toujours exactement au centre de sa voie et notre base de données ne comporte aucune image où l'agent commence à dévier de sa voie ou se trouve en situation d'échec. C'est pour cette raison que pour nos premiers essais les performances de l'agent était très mauvaises. En effet, nous avons remarqué que dès que l'agent commençait à dévier de sa voie, les cartes de segmentations estimées étaient très mauvaises et comme les poids de l'encodeur sont fixés, ces performances ne pouvaient pas s'améliorer au cours de l'apprentissage. Il fallait en fait entraîner notre encodeur à être robuste aux erreurs que l'agent va nécessairement commettre lorsqu'il va explorer l'environnement. D'ailleurs cela montre bien que les estimations auxiliaires peuvent permettre dans certains cas de pouvoir expliquer les décisions prises par l'agent, nous avons trouvé ce problème en observant explicitement les estimations obtenues par notre décodeur durant l'entraînement par renforcement de l'agent. L'intuition de ce problème est illustrée sur la Figure 5.15. Pour résoudre ce problème, nous suggérons d'ajouter des augmentations de point de vue en déplaçant la caméra aléatoirement autour de la vraie position de l'autopilote. Avec ces augmentations la performance de l'encodeur était bien meilleure lorsque l'agent conduisait et explorait, et nous avons trouvé que c'était nécessaire pour obtenir des bonnes performances pendant la phase d'apprentissage par renforcement.

Pour résumer, pour pouvoir utiliser un grand réseau avec des tailles d'images importantes, nous suggérons de pré-entraîner un encodeur avec des tâches supervisées bien choisies, puis d'utiliser les features finales de l'encodeur, les *indices implicites*, comme espace d'état pour l'entraînement par renforcement. Il faut aussi que la base de données utilisée pour l'entraînement de l'encodeur soit en quelque sorte bruitée afin d'être sûr que l'encodeur soit adapté durant l'exploration de l'agent.

5.5.2 Gérer la discrétisation des actions

Comme nous l'avons mentionné plusieurs fois, les algorithmes de RL fondés sur la fonction de valeur comme DQN, Rainbow ou bien Rainbow-IQN [Mnih et al., 2015, Hessel et al., 2018, Toromanoff et al., 2019] nécessitent de passer par des actions discrètes. Dans nos expériences préliminaires avec peu d'actions possibles (seulement 5 actions différentes pour l'angle au volant), les agents oscillaient énormément et n'arrivaient pas à rester sur leur voie. Nous avons obtenu de meilleures performances en augmentant simplement le

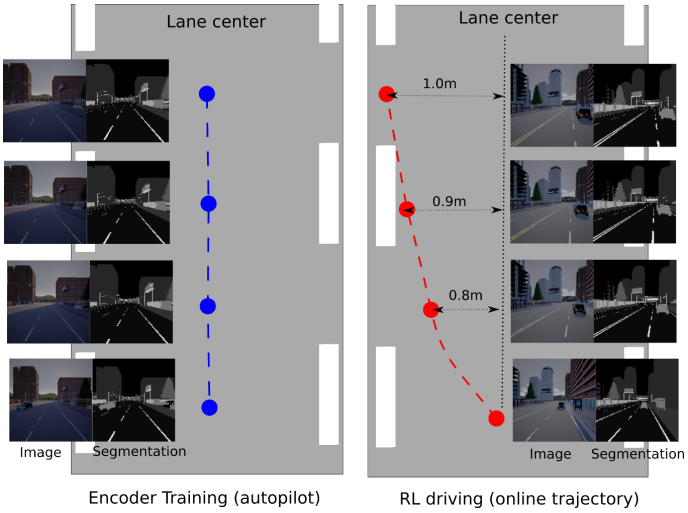


FIGURE 5.15 – Illustration de pourquoi il est nécessaire de rajouter des points de vue différents pour l'entraînement de l'encodeur : les trajectoires de l'agent en train d'apprendre par renforcement (à droite) peuvent dévier du centre de la voie. Cela résulte en des cartes de segmentation avec beaucoup plus de variabilité, particulièrement pour les positions des marquages au sol, que celles obtenues si on entraîne seulement depuis des enregistrement du pilote automatique de CARLA (à gauche).

nombre d'actions disponibles : nous sommes passé de 5 angles au volant possibles à 9 ou 27 valeurs différentes. L'accélération était moins problématique et nous avons utilisé seulement 3 valeurs différentes d'accélération plus une pour freiner. Dans la majorité de nos expériences il y avait donc un total de 36 (9 x 4) ou bien 108 (27 x 4) actions. Nous avons aussi essayé de calculer la dérivée de l'angle au volant : la sortie du réseau est utilisée pour mettre à jour l'angle au volant précédent (qui est donné en entrée) au lieu de prendre directement la sortie comme angle au volant courant. L'impact de ces différents choix est étudié dans la section 5.6.

Afin d'améliorer les performances finales et de réduire les oscillations, nous suggérons fortement d'utiliser un ensemble de plusieurs agents et de moyenner leurs sorties. Pour cela, nous avons simplement pris les poids successifs enregistrés périodiquement au cours d'un même entraînement : cela est accessible très facilement et évite d'avoir à effectuer plusieurs entraînements. Cette astuce améliore systématiquement le comportement de l'agent en réduisant considérablement les oscillations et permet au final d'obtenir des meilleures performances. Par ailleurs, comme les poids de l'encodeur sont fixés, son temps d'inférence peut être partagé : le temps de calcul supplémentaire rajouté lorsque l'on moyenne plusieurs réseaux issus d'un même entraînement est négligeable (moins de 10% du temps d'inférence total lorsque l'on moyenne les sorties de trois agents). C'est la raison pour laquelle nous avons toujours utilisé cette astuce pour évaluer les performances de nos agents. En pratique nous moyennons les sorties de trois agents séparés de 1 million de pas d'entraînement, c'est à dire que les résultats à 10 millions de pas

sont obtenus en utilisant la moyenne des sorties des trois réseaux à 8, 9 et 10 millions de pas d'entraînement. Cette approche est d'ailleurs souvent utilisée pour le contrôle, même pour des valeurs continues. Les travaux décrits dans le chapitre suivant, Chapitre 6, ont d'ailleurs prouvé l'utilité de cette méthode.

Pour résumer, les actions discrètes peuvent être compensées en utilisant une discrétisation suffisamment fine des actions et surtout en moyennant les sorties (discrètes) de plusieurs agents.

5.6 Expériences et études d'ablations

5.6.1 Introduction d'une situation de test et d'une métrique pour une évaluation commune

Afin de pouvoir évaluer et comparer nos différentes expériences et nos études d'ablations, nous avons défini un ensemble de scénarios communs ainsi qu'une métrique associée pour faire des comparaisons équitables. En effet, au moment de la publication de notre article, l'environnement et les résultats du challenge CARLA n'étaient pas publics. De plus le benchmark CARLA n'était disponible que sur une version obsolète de CARLA (0.8.X) sur laquelle le rendu et la physique diffèrent de la version de CARLA du challenge (0.9.X), ce n'était donc pas adapté pour notre situation. Surtout que, comme susmentionné, le benchmark CARLA est une tâche bien plus simple que le challenge CARLA. Nous verrons en fin de section, que nous avons quand même pu évaluer notre agent sur le benchmark CARLA mis à jour sur la nouvelle version de CARLA.

Définir des scénarios d'évaluation commun Nous avons choisi d'utiliser l'environnement le plus difficile dans les villes de CARLA : *Town05*. Cette carte inclut la ville la plus grande, possède principalement des routes avec plusieurs voies et est américaine : les feux de signalisation sont à l'opposé de l'intersection et sont donc beaucoup plus difficiles à détecter. Nous faisons aussi apparaître de temps en temps des piétons traversant la route devant l'agent pour vérifier que nos agents freinent dans ces situations. Finalement, nous changeons les conditions de luminosité et météo (pluie, soleil, nuages) pour rendre la tâche la plus difficile possible. De cette manière, même avec un entraînement sur une seule ville, nous avons une tâche très complexe. L'entraînement sur une seule ville est nécessaire pour pouvoir faire nos expériences et nos études d'ablations en un temps raisonnable même si cela rend la tâche bien moins complexe que le challenge CARLA où l'évaluation se fait dans des villes que l'agent n'a jamais vu durant l'entraînement.

Toutes nos expériences ont été effectuées avec 20 millions d'itérations dans CARLA, avec 3 acteurs (donc 6.6 millions de pas pour chacun) avec 10 images par seconde. Ainsi 20 millions d'images correspond à environ 20 jours de temps simulé : à titre de comparaison la majorité des algorithmes de renforcement évalués sur Atari [Mnih et al., 2015, Fortunato et al., 2017], utilisent 200 millions d'images ce qui correspond à 40 jours de temps simulé, certains utilisent même plus de 5 années [Kapturowski et al., 2018,

Horgan et al., 2018]) de temps simulé.

Nous avons défini 10 scénarios dans l'environnement urbain de *Town05* qui consistent chacun en 10 intersections successives et 10 commandes haut niveau associées indiquant la voie à suivre à chaque intersection. Nous avons aussi défini quelque scénarios sur la partie autoroute de Town05 mais ces scénarios étaient beaucoup plus simples et ainsi bien moins discriminants : par exemple notre meilleur agent sortait de sa voie moins d'une fois tous les 100km sur des situations d'autoroute. Ces scénarios plus simples d'autoroute ont été principalement utilisés pour mesurer l'oscillation de nos différents agents.

Définir une métrique pour comparer différents modèles Nous avons évalué chacun de nos modèles 10 fois sur chaque scénario en faisant varier les positions initiales des autres agents ainsi que les conditions de luminosité et météo. Contrairement à la phase d'entraînement, nous terminons un épisode seulement quand l'agent sors de sa voie car cela permet de compter le nombre d'infractions effectuées par l'agent. Notre métrique principale est le pourcentage d'intersections que l'agent a réussit à franchir (*Inters.*, plus grand étant meilleur), par exemple 50% d'achèvement correspond à une moyenne de 5 intersections franchies dans chaque scénario. Nous enregistrons aussi le pourcentage de feux tricolores passés sans infraction (*Feu*, plus grand étant meilleur) ainsi que le pourcentage des piétons apparus devant l'agent où l'agent a bien freiné sans faire de collision (*Piétons*, plus grand étant meilleur). Il est important de noter que les 2 dernières métriques sont moins pertinentes car un agent qui ne bougerait jamais ne grillera jamais un feu rouge et ne fera pas non plus de collisions. C'est pourquoi la métrique *Inters.* est notre métrique principale pour faire des comparaisons : *Feu* et *Piéton* sont utilisés pour une comparaison plus fine. Nous avons aussi introduit une métrique pour mesurer les oscillations : la moyenne de la rotation entre l'agent et la route au long de l'épisode (*Osc.*, plus petit est meilleur).

5.6.2 Études d'ablations sur la phase d'entraînement supervisé de l'encodeur

Dans cette section, nous détaillerons nos études d'ablations sur la phase d'apprentissage supervisé des *indices*. La deuxième phase d'apprentissage par renforcement est exactement identique pour avoir des comparaisons équitables.

Tout d'abord nous avons effectué quelques expériences en ne faisant pas du tout la phase d'apprentissage supervisé, c'est à dire que nous avons entraîné le réseau entier à partir de zéro durant l'apprentissage par renforcement. Nous avons comparé 3 architectures différentes : d'abord le réseau initial de DQN [Mnih et al., 2015] avec des images en niveau de gris 84x84, puis une amélioration très simple du réseau de DQN (*Plus grand DQN*) qui prend en entrée des images de taille 288x288x3 et finalement notre modèle final avec l'encodeur Resnet-18 [He et al., 2016].

La Figure 5.16 montre que sans la phase d'apprentissage des *indices*, les agents n'arrivent pas à apprendre et ne réussissent même pas à franchir une intersection en moyenne (la métrique *Inters.* est à moins de 10%). De plus, il est important de noter qu'entraî-

ner le *Plus grand DQN* (respectivement le Resnet-18) a nécessité 50% (respectivement 200%) plus de temps qu'entraîner la même architecture avec notre stratégie d'*indices implicites* même en prenant en compte le temps nécessaire à la phase d'apprentissage supervisé. En raison de leur temps d'apprentissage beaucoup plus long et de leur performance extrêmement faible, nous avons arrêté ces expériences à seulement 10 millions de pas d'apprentissage. Ces entraînements ont aussi nécessité beaucoup plus de mémoire car les images brutes devaient être enregistrées dans la mémoire de reprise. Comme prévu, ces expériences ont montré à quel point il est difficile d'entraîner un grand réseau en utilisant seulement le signal d'apprentissage du renforcement.

FIGURE 5.16 – Évolution des performances (métrique *Inters.*) de différents choix d'encodeur en fonction du nombre d'itérations. Le premier groupe d'agents (trait plein) ont des encodeurs entraînés avec différentes fonctions de coût et dont les poids sont fixés durant l'apprentissage par renforcement. Le second groupe (pointillé) ont des encodeurs qui ne sont entraînés que par renforcement (arrêtés plus tôt car leurs performances sont clairement moins bonnes).

La deuxième partie des études d'ablations concernait l'entraînement de notre encodeur Resnet-18. Premièrement, en tant que garde-fou, nous avons utilisé un encodeur sans l'entraîner, i.e. notre encodeur est fixé avec des poids aléatoires. Nous avons ensuite retiré certains *indices* de l'apprentissage supervisé pour étudier leur impact sur les performances finales : une expérience sans utiliser l'estimation de l'état des feux tricolores, et une expérience sans l'estimation de la carte de segmentation. La Figure 5.16 montre que ces deux ablations atteignent des performances bien moindres que notre expérience standard : cela montre bien l'intérêt d'estimer l'état des feux ainsi que la segmentation sémantique durant l'apprentissage supervisé.

La Table 5.1 montre qu'enlever l'estimation de l'état du feu à un grand impact sur les performances finales. Comme prévu, l'agent qui utilise un encodeur dont le décodeur n'a pas appris à estimer explicitement l'état des feux va moins souvent passer au feu vert (métrique *Feu* à 80% au lieu de 97.6%). Cependant, il est intéressant de noter que la performance de l'agent est bien meilleure qu'un choix aléatoire qui serait de seulement

Encodeur utilisé	Inters.	Feu	Piéton
Aléatoire	0%	NA	NA
Sans état du feu	33.4%	80%	82%
Sans segmentation	41.6%	96.5%	63%
Tous les indices	61.9%	97.6%	76%

TABLE 5.1 – Comparaison de la performance des agents utilisant différents pré-entraînements de l'encodeur : un encodeur avec des poids aléatoires, un encodeur entraîné sans l'estimation par le décodeur de l'état des feux, un encodeur entraîné sans l'estimation de la carte sémantique, et finalement l'encodeur entraîné avec tous les indices illustrés en Figure 5.14.

25% car les feux ne sont verts que 25% du temps. Cela signifie que l'agent réussit quand même à détecter l'état des feux dans les features de l'encodeur. Nous supposons que comme la segmentation sémantique inclut une classe pour les feux tricolores (mais par leur état), les caractéristiques contiennent malgré tout de l'information sur l'état des feux. L'agent pourrait aussi avoir appris à observer les autres véhicules et comprendre qu'il faut avancer en même temps que les véhicules environnants.

Enlever l'estimation de la segmentation sémantique a aussi un impact important sur les performances finales. Il est intéressant de noter que cet agent a la pire performance *Piéton* ce qui signifie que l'agent à du mal à détecter les piétons et les véhicules : cette information n'est en effet contenue que dans la carte de segmentation.

5.6.3 Études d'ablations sur la phase d'entraînement par renforcement

Pour faire des comparaisons équitables, le même encodeur pré-entraîné a été utilisé pour toutes les expériences qui vont suivre, entraîné avec tous les *indices* mentionnées sur la Figure 5.14. L'encodeur utilisé est en fait celui qui nous a servi pour le challenge CARLA, entraîné sur beaucoup plus de données (provenant de plusieurs villes de CARLA) et pendant plus de temps que les encodeurs utilisés pour les études d'ablations décrites précédemment.

Nous avons fait deux études d'ablations sur la fonction de récompense. Dans la première (*vitesse désirée constante*), la vitesse désirée n'est plus adaptée à la situation : l'agent doit comprendre comment s'arrêter aux feux rouges et éviter les collisions seulement depuis le signal de fin d'épisode. Dans la deuxième expérience, nous avons retiré le terme *rotation désirée* de la fonction de récompense pour évaluer l'impact sur les oscillations de l'agent. Nous avons aussi expérimenté sur l'espace d'actions de l'agent. Nous avons d'abord fait calculer la dérivée de l'angle au volant à l'agent au lieu de l'angle courant. Finalement, nous avons modifié le nombre d'actions d'angle au volant disponibles en testant soit 9 valeurs différentes soit 27. Les résultats de toutes ces expériences sont présentés dans la Table 5.2.

Le résultat le plus intéressant de ces expériences est celui avec la *vitesse désirée constante*. En effet, on constate que l'agent échoue totalement à s'arrêter, que ce soit pour les feux rouges ou bien pour éviter les collisions : ses performances de *Feu* et *Piéton*

Expérience	Inters.	Feu	Piéton	Osc.
Vitesse désirée constante	50.3%	31%	42%	1.51°
Sans rotation désirée	64.7%	99%	77.7%	1.39°
27 valeurs d'angle (dérivée)	64.5%	98.7%	85.1%	1.64°
9 valeurs d'angle (absolu)	74.4%	98.5%	84.6%	0.88°
27 valeurs d'angle (absolu)	75.8%	98.3%	81.6%	0.84°

TABLE 5.2 – Comparaison de la performance des agents selon la fonction de récompense utilisée ainsi que de l'espace d'action. Une expérience sans adaptation de la *vitesse désirée* et une sans récompense associée à la *rotation désirée*. Concernant l'espace d'action, nous avons expérimenté de faire calculer à l'agent la dérivée de l'angle au volant au lieu de l'angle au courant. Finalement, nous avons comparé entre 9 et 27 valeurs d'angle au volant absolus.

sont bien plus mauvaises que les autres agents. L'agent entraîné avec une vitesse désirée constante grille plus de 70% des feux ce qui est très proche d'un choix aléatoire ! De plus, il percute presque 60% des piétons que l'on fait apparaître devant lui. Cette expérience montre à quel point le terme *vitesse désirée* de notre récompense est important afin d'apprendre comment freiner.

De manière surprenante, nous avons observé que calculer la dérivée de l'angle au volant augmentait les oscillations de l'agent, encore plus que quand nous avons retiré le terme *rotation désirée* de notre récompense. Finalement, prendre 9 ou 27 actions différentes d'angle au volant n'a pas d'impact significatif et ces 2 agents atteignent les meilleures performances avec le moins d'oscillations.

5.6.4 Étude de la généralisation sur des villes jamais vues

Finalement, nous avons fait aussi quelques expériences sur la généralisation, dans l'optique d'être le plus proche possible du challenge CARLA. Pour cela, nous avons entraîné un agent sur 3 villes différentes en même temps (une européenne et deux américaines) et évalué cet agent sur 2 villes jamais vues, une américaine et une européenne. Nous avons evalué notre meilleur agent appris sur seulement *Town05* pour avoir une référence.

Entraînement	Ville EU jamais vue	Ville US jamais vue
Seulement Town05	2.4%	42.6%
Plusieurs villes	58.4%	36.2%

TABLE 5.3 – Comparaison des performances (métrique *Inters.*) entre notre meilleur agent entraîné seulement sur *Town05* et un agent entraîné sur plusieurs villes différentes en même temps. L'agent n'ayant jamais été entraîné sur des villes européennes échoue totalement lorsqu'il est testé sur ce type d'environnement.

Les résultats sont présentés dans la Table 5.3. Nous pouvons observer que les performances de l'agent entraîné seulement sur une ville américaine sont très faibles sur

la ville européenne jamais vue. Cela confirme bien l'intérêt d'entraîner l'agent sur des villes européennes et américaines en même temps. Sur la ville américaine jamais vue, les performances sont à peu près similaires pour les deux entraînements. Ces expériences montrent bien que notre approche généralise à des environnements que l'agent n'a jamais vus du moment qu'il y a eu suffisamment de variété dans les données d'entraînement.

5.6.5 Comparaison sur le benchmark CARLA

Après la soumission de notre article à CVPR, Chen et al. [2020a] ont ré-implémenté le benchmark CARLA sur la version la plus récente de CARLA et les chercheurs ont mis cette implémentation en libre accès. Nous avons ainsi pu ré-entrainer et évaluer notre méthode sur cette nouvelle version du benchmark CARLA. Les résultats de ces expériences sont présentés Table 5.4.

	ville d'entraînement					ville de test				
Tâche (météo entraînement)	RL	CAL	CILRS	LBC	Ours	RL	CAL	CILRS	LBC	Ours
Tout droit	89	100	96	100	100	74	93	96	100	100
Un virage	34	97	92	100	100	12	82	84	100	100
Navigation	14	92	95	100	100	3	70	69	98	100
Nav. dynamique	7	83	92	100	100	2	64	66	99	98

	ville d'entraînement					ville de test				
Tâche (météo test)	RL	CAL	CILRS	LBC	Ours	RL	CAL	CILRS	LBC	Ours
Tout droit	86	100	96	100	100	68	94	96	100	100
Un virage	16	96	96	96	100	20	72	92	100	100
Navigation	2	90	96	100	100	6	88	92	100	100
Nav. dynamique	2	82	96	96	100	4	64	90	100	100

TABLE 5.4 – Comparaison des taux de succès (en % pour chaque tâche et scénario, plus grand étant meilleur) avec les références RL [Dosovitskiy et al., 2017], CAL [Sauer et al., 2018], CILRS [Codevilla et al., 2019] et LBC [Chen et al., 2020a] sur le benchmark CARLA. Au dessus, le tableau des résultats avec les conditions météo de l'entraînement. En bas, les résultats avec des conditions météo jamais vues durant l'entraînement.

Ainsi nous observons que notre approche atteint quasiment les performances maximales sur le benchmark CARLA initial et performe bien mieux que la majorité des références de l'état de l'art : seul l'article LBC [Chen et al., 2020a] atteint des performances comparables aux nôtres. Il est important de noter que toutes les autres références utilisent de l'apprentissage par imitation. Nous détaillerons ces approches dans le chapitre suivant, Chapitre 6. Seule la référence *RL* [Dosovitskiy et al., 2017] fait de l'apprentissage par renforcement mais cette référence a des performances beaucoup plus faibles que toutes les autres références. Cette expérience a permis de montrer que notre approche par renforcement était bien comparable voir même meilleure que l'état de l'art actuel. De plus, c'est la première fois qu'une approche par renforcement réussit à atteindre des performances meilleures que des approches utilisant de l'apprentissage par imitation.

Le benchmark *NoCrash* Les chercheurs de l'article LBC [Chen et al., 2020a] ont aussi développé un autre benchmark (appelé *NoCrash*) beaucoup plus difficile que le benchmark initial. En effet, dans cette deuxième version, les chemins à parcourir sont beaucoup plus longs et l'agent n'a pas le droit de faire de collision. De plus, il existe 3 catégories, *vide*, *trafic normal* et *trafic dense*, qui font varier le nombre de véhicules et piétons présents dans la ville. Il peut être noté que les catégories *trafic normal* et surtout *trafic dense* sont tellement difficiles que même le pilote automatique de CARLA ne réussit pas à atteindre les 100% de succès. Les résultats de ces expériences sont présentés Table 5.5 pour le benchmark *NoCrash*.

	ville d'entraînement		ville de test	
Tâche (météo entraînement)	LBC	Ours	LBC	Ours
Vide	97	100	100	99
Trafic normal	93	96	94	87
Trafic dense	71	70	51	42

	ville d'entraînement		ville de test	
Tâche (météo test)	LBC	Ours	LBC	Ours
Vide	87	36	70	24
Trafic normal	87	34	62	34
Trafic dense	63	26	39	18

TABLE 5.5 – Comparaison des taux de succès (en % pour chaque tâche et scénario, plus grand étant meilleur) avec les références [Chen et al., 2020a] sur le benchmark *NoCrash*. Au dessus le tableau des résultats avec les conditions météo de l'entraînement, en bas les résultats avec des conditions météo jamais vues durant l'entraînement.

Sur le récent benchmark *NoCrash*, nous avons des résultats très similaires à la référence *LBC* [Chen et al., 2020a] sous les conditions météo d'entraînement. A noter que l'on ne peut se comparer qu'à cette approche car toutes les autres n'ont pas été évaluées sur ce benchmark beaucoup plus dur. Il est cependant légitime de considérer que comme nos performances étaient bien meilleures sur le benchmark initial, elles seraient sûrement meilleures sur le benchmark *NoCrash*.

Par contre, nous avons observé que notre agent avait beaucoup de problèmes à généraliser des conditions météo d'entraînements aux conditions météo d'évaluation. Nous avons découvert que notre encodeur (fixé durant l'entraînement par renforcement) confondait les reflets du soleil avec des "objets mobiles" : ces reflets du soleil sur le sol n'étaient jamais présents dans l'entraînement. Cela explique pourquoi la grande majorité de nos cas d'échec durant l'évaluation sous les conditions météo jamais vues était des *timeout*, i.e. notre agent ne bougeait plus jusqu'à la fin du temps imparti. En effet dès qu'il y avait une tache de soleil sur le sol devant lui, l'agent agissait comme si un véhicule ou un piéton était devant et s'arrêtait pour toujours. Réussir à gérer diverses conditions de luminosité est un problème récurrent des algorithmes de perception et nous pensons qu'améliorer notre phase d'entraînement supervisé (particulièrement la segmentation sémantique) permettrait probablement de gérer ce problème.

5.7 Conclusion

Dans ce travail, nous avons présenté le premier agent entraîné par renforcement réussissant à conduire dans un environnement urbain complexe à partir uniquement d'images caméras. Nous avons utilisé un algorithme de renforcement distribué, Rainbow-IQN Ape-X, entrainé avec une fonction de récompense adaptée et un réseau de neurones conditionnel possédant bien plus de paramètres que les réseaux utilisés communément en DRL. Pour réussir à faire fonctionner un apprentissage par renforcement dans ce contexte très complexe, nous avons introduit la méthode d'*indices implicites*, qui entraîne de manière supervisée un encodeur-décodeur à estimer des informations sémantiques utiles pour la conduite en ville. Nous avons validé nos choix techniques en effectuant plusieurs études d'ablations et finalement nous avons montré la performance de notre méthode en gagnant le challenge CARLA en 2019 sur la catégorie "Cameras Only". Nous avons aussi remporté le challenge CARLA en 2020 sur la catégorie "All Sensors".

Chapitre 6

Intégration sur véhicule réel : Apprentissage par imitation

Les résultats de ce chapitre ont été présentés au CES 2018 à Las Vegas pour une démonstration sur un vrai véhicule.

Les contributions de ce chapitre ont aussi été publiées dans [Toromanoff et al., 2018] :

Marin Toromanoff, Emilie Wirbel, Frédéric Wilhelm, Camilo Vejarano, Xavier Perrotton, and Fabien Moutarde. End to end vehicle lateral control using a single fisheye camera. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3613–3619. IEEE, 2018.

6.1 Introduction

Tout au long de ce manuscrit, nous avons toujours appliqué nos algorithmes de renforcement en simulation. Que ce soit dans le Chapitre 3 sur des jeux de courses, dans le Chapitre 4 sur un émulateur de jeu Atari et surtout dans le chapitre précédent, Chapitre 5, où l'on a appliqué notre algorithme à de la conduite urbaine dans le simulateur CARLA. Dans les deux derniers chapitres de ce manuscrit, nous nous intéresserons spécifiquement à l'intégration sur un véhicule réel.

Dans un premier temps, nous présenterons un travail effectué au début de cette thèse, le contrôle latéral d'un véhicule réel pour du maintien de voie, présenté à la conférence IROS 2018 [Toromanoff et al., 2018]. Nous verrons que nous n'avons pas utilisé d'apprentissage par renforcement mais de l'apprentissage par imitation pour ce travail. L'idée est que l'apprentissage par imitation est généralement plus simple à effectuer car il peut se mettre sous la forme d'un apprentissage supervisé et cela est un bon point de départ pour intégrer un apprentissage de bout-en-bout sur un véhicule réel.

Nous allons donc d'abord faire une description détaillée de l'apprentissage par imitation ainsi que dresser un état de l'art de l'apprentissage par imitation pour la voiture autonome. Nous détaillerons plus particulièrement les travaux d'imitation sur CARLA [Chen et al., 2020a, Codevilla et al., 2018] utilisés comme référence dans le chapitre précédent ainsi que certains articles représentatifs d'imitation appliqués sur une voiture réelle. Finalement, nous présenterons notre article publié à la conférence IROS 2018 intitulé : *End to end lateral control using a single fisheye camera* [Toromanoff et al., 2018].

6.1.1 Présentation de l'apprentissage par imitation

L'apprentissage par imitation poursuit le même objectif final que l'apprentissage par renforcement, i.e. apprendre une politique capable de résoudre une tâche spécifique. On reste donc dans le cas d'un processus de décision markovien comme pour l'apprentissage par renforcement, c.f. Chapitre 2. La différence fondamentale est qu'on ne laisse plus l'agent apprendre par lui-même à partir d'une fonction de récompense mais on lui fait apprendre à partir de démonstrations expertes, i.e. les capteurs et les actions prises par un conducteur humain dans le cas de la voiture autonome.

Il y a en fait plusieurs manières d'apprendre une politique à partir de ces données expertes, la plus simple et la plus intuitive est appelée *réplication du comportement* (*behavorial cloning* en anglais). La réplication du comportement ramène l'apprentissage d'une politique à un apprentissage supervisé avec les entrées étant les états observés par l'expert et les sorties étant les actions prises par cet expert.

Une autre méthode d'apprentissage par imitation est l'*apprentissage par renforcement inverse* [Ng et al., 2000]. Cette méthode consiste à essayer de déterminer une fonction de récompense telle qu'un apprentissage par renforcement avec cette récompense produit une politique qui se rapproche le plus possible des trajectoires expertes initiales. Cependant, cette technique est relativement peu mature et n'a pour l'instant jamais été appliquée avec succès dans le cadre du contrôle d'une voiture autonome dans

un environnement complexe comme CARLA. Pour cette raison, nous nous intéresserons principalement à l'apprentissage par réplication du comportement dans tous ce chapitre.

Il y a cependant des problèmes inhérents à l'apprentissage par imitation. Tout d'abord, comme l'on cherche à imiter exactement les actions prises par l'expert, il n'y a pas d'amélioration possible par rapport à cet expert : au mieux on arrive aux mêmes performances. De plus, il est nécessaire d'avoir initialement de nombreuses trajectoires d'un tel expert. Dans le cadre de la voiture autonome, il est en fait assez simple de collecter ce genre de trajectoires, il suffit juste d'enregistrer les actions (angle au volant, pédales d'accélération et de frein) et les entrées des capteurs pendant qu'un humain conduit le véhicule.

En fait, le véritable problème de l'apprentissage par imitation est ce que l'on appelle la *non-concordance des distributions* (*distribution mismatch* en anglais) entre la distribution des données de l'entraînement et celle obtenue lors de la phase d'évaluation. En effet, les états observés par l'agent durant la phase d'apprentissage sont des états issus de trajectoires expertes, il n'y a donc jamais de situation d'échec. L'agent ne peut donc pas apprendre quoi faire dans ce type de situation car ces situations n'auront jamais été présentes dans la base d'entraînement. Cependant, lorsque l'agent est évalué sur la tâche, il effectue nécessairement des petites erreurs. Ces erreurs en s'accumulant vont éloigner petit à petit l'agent de la trajectoire optimale pour finalement l'amener dans des situations trop loin de la distribution des données expertes où l'agent ne saura pas comment réagir. Ce problème est illustré sur la Figure 6.1. La non-concordance de distributions est aussi extrêmement corrélée au problème d'évaluation en *boucle ouverte* contre évaluation en *boucle fermée*. Lorsque l'agent est évalué en boucle ouverte, cela signifie que le contrôle choisi par l'agent n'est pas réellement appliqué, on calcule juste une fonction de coût (comme l'erreur quadratique moyenne) dépendant de la différence instantanée entre le contrôle choisit par l'agent et le contrôle qu'avait pris l'expert. Au contraire, lorsque l'on évalue l'agent en boucle fermée, l'action prise par l'agent est effectuée et on peut ainsi évaluer comment l'agent réagit à ses propres erreurs. Nous allons voir par la suite, que l'évaluation en boucle ouverte est moins voire parfois pas du tout significative : un agent peut avoir une fonction de coût extrêmement faible durant l'apprentissage par imitation en boucle ouverte et pourtant échouer totalement à la tâche lorsqu'il est évalué en boucle fermée. Il faut donc nécessairement évaluer les agents appris par imitation en boucle fermée pour avoir une idée des performances réelles.

Ce problème de non-concordance de distributions est très important pour l'apprentissage par imitation. Par exemple, nous verrons prochainement, c.f. section 6.1.2, que si l'on apprend simplement à imiter l'angle au volant à partir d'une base de données d'un conducteur humain, l'agent sera incapable de se maintenir sur sa voie en phase d'évaluation. L'idée fondamentale est donc d'introduire dans la base de données d'apprentissage des situations d'échec afin d'apprendre à l'agent à réagir à ces situations et revenir à une situation optimale. L'article DAgger (pour *Dataset Aggregation*) [Ross et al., 2011] présente une idée simple pour pallier ce problème de distributions différentes entre l'entraînement et l'évaluation. L'idée est de rajouter à la base de données d'entraînement les expériences obtenues lors de l'évaluation de l'agent en particulier les situations d'échecs. Ensuite, un expert annote chacune de ces nouvelles expériences avec l'action qui devrait

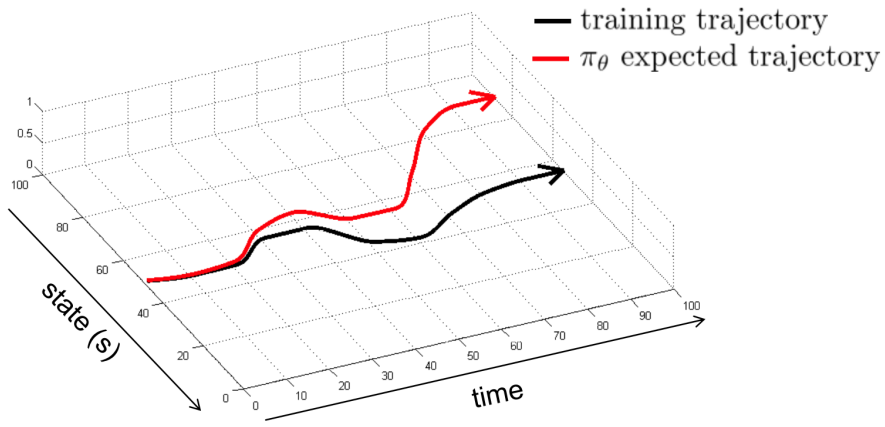


FIGURE 6.1 – Illustration de la *non-concordance de distributions* entre la distribution des trajectoires expertes et celle obtenue lors de l'évaluation de l'agent entraîné par imitation. Initialement, on voit que la trajectoire de l'agent (en rouge) est très proche de la trajectoire d'entraînement (en noir). Pourtant, en accumulant des petites erreurs, l'agent s'éloigne après quelques itérations de la trajectoire experte pour finalement arriver dans des états qu'il n'a jamais vus et où il ne sait pas comment réagir. Image issue de Levine [2017].

être prise. Puis on ré-entraîne un agent par imitation sur cette nouvelle base de données composée de l'ensemble entre les données expertes initiales et les données de l'agent annotées par l'expert. On peut ensuite répéter ce processus le nombre de fois nécessaire et s'arrêter seulement lorsque les performances de l'agent sont suffisamment bonnes. Cet algorithme est présenté sur la Figure 6.2.

1. train $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{a}_1, \ldots, \mathbf{o}_N, \mathbf{a}_N\}$
2. run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions $\mathbf{a}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

FIGURE 6.2 – Algorithme de DAgger (*Dataset Aggregation*) [Ross et al., 2011]. L'idée est de rajouter à la base de données initiale des observations obtenues lors de l'évaluation de l'agent. Ces observations sont annotées par un expert indiquant quelle est l'action qui devrait être prise. Cela permet à l'agent de savoir que faire lors des situations d'échec et comment revenir à une situation optimale. Image issue de Levine [2017].

Il est important de noter que l'algorithme DAgger nécessite de pouvoir appliquer dans l'environnement réel, un agent qui peut avoir des performances extrêmement faibles. En effet à la première itération, l'agent qui n'aura jamais observé de situation d'échec peut avoir un comportement en boucle fermée extrêmement mauvais. Dans une grande partie des applications, il n'est pas possible d'agir de la sorte pour des raisons de sécurité, par

exemple par risque d'accident de la route pour la voiture autonome. De plus, cet algorithme nécessite qu'un expert puisse annoter automatiquement les situations d'échec que l'agent a rencontré lors de l'évaluation. Cette annotation peut se révéler extrêmement coûteuse voire parfois impossible. Pour ces deux raisons, l'algorithme DAgger n'est en fait généralement pas utilisable en pratique, et en particulier semble très compliqué à appliquer à la voiture autonome.

Il existe un autre problème très important à l'apprentissage par imitation, la *confusion de causalité*. Ce problème est en particulier expliqué dans l'article de de Haan et al. [2019]. Pour illustrer ce problème les chercheurs donnent l'exemple où un agent doit décider si il faut freiner ou non. Dans le premier cas, l'agent peut voir toutes les informations d'une caméra située derrière le conducteur humain, c'est à dire le tableau de bord du véhicule ainsi que voir à travers le pare-brise. Dans le deuxième cas, le tableau de bord est caché et l'agent n'a accès qu'à la partie de l'image qui voit à travers le pare-brise. Il y a donc strictement plus d'information fournie à l'agent dans le premier cas et les performances en boucle ouverte seront bien meilleures dans le premier cas. Pourtant, lors de l'évaluation des deux agents, le premier échouera totalement tandis que le deuxième réussira à freiner dans les bonnes situations. Cela est dû au fait que dans le premier cas, l'agent ne regarde que l'indicateur du tableau de bord pour savoir si il doit freiner, ce faisant il ne fera une erreur qu'à l'instant exact où l'humain a commencé à freiner. Dans le deuxième cas, l'agent doit nécessairement apprendre à utiliser les informations pertinentes pour savoir si il faut freiner ou non, comme la présence d'un piéton devant le véhicule. Ce problème est illustré sur la Figure 6.3.

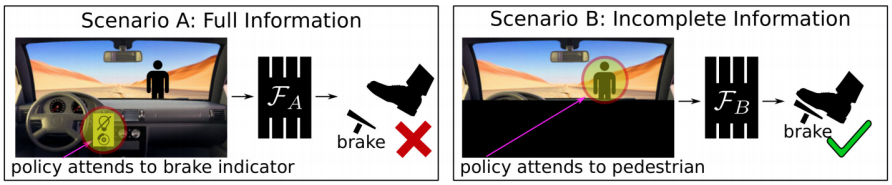


FIGURE 6.3 – Un autre problème de l'apprentissage par imitation : la *confusion de causalité*. Dans le premier cas, l'agent utilise uniquement le signal indicateur du frein sur le tableau de bord pour définir si il faut freiner. Ce faisant, il ne fera une erreur qu'à l'exact moment où le conducteur humain commence à freiner. Cependant, lorsque cet agent sera évalué en boucle fermée, il ne freinera jamais car l'indicateur du tableau de bord ne sera jamais allumé et l'agent échouera totalement à la tâche. Dans le deuxième cas, l'agent doit nécessairement apprendre les informations pertinentes comme la présence ou non de piéton pour savoir quand freiner. Image issue de de Haan et al. [2019].

Ce problème montre à quel point les métriques en boucle ouverte peuvent être totalement décorrélées de la performance finale. Il faut en fait, toujours évaluer les agents appris par imitation en boucle fermée pour réellement savoir si les performances sont bonnes ou non. Il est important de noter que ce problème de *confusion de causalité* ne survient pas lorsque l'on entraîne un agent par renforcement, en effet en renforcement

l'entraînement et l'évaluation se font en boucle fermée.

6.1.2 L'apprentissage par imitation appliqué à la voiture autonome

Nous allons maintenant voir les travaux d'apprentissage par imitation appliqués spécifiquement à la voiture autonome.

6.1.2.1 Sur un véhicule réel

L'article *End to End Learning for Self-Driving Cars* [Bojarski et al., 2016] est le premier travail à avoir réussi à appliquer de l'apprentissage par imitation sur une voiture réelle pour faire du maintien de voie. Il est important de noter que ce travail s'est grandement inspiré des idées introduites dans un travail précurseur de la DARPA [Pomerleau, 1989] effectué plus de 15 années auparavant !

Pour cela les chercheurs de Nvidia ont utilisé l'image d'une caméra frontale et ont entraîné un réseau de neurones à calculer les mêmes angles au volant que l'expert sur une base de données comportant 72 heures de conduite. Afin de pallier au problème de non-concordance des distributions décrit précédemment, Bojarski et al. [2016] ont utilisé trois caméras frontales (une au milieu du pare-choc avant, les deux autres décalées à gauche et à droite du pare-choc) pour faire de *l'augmentation de labels*. En effet, dans le cas du maintien de voie, les situations d'échec sont en fait les cas où la voiture est soit trop à droite, soit trop à gauche de la voie. L'idée est ainsi de rajouter à la base de données originelle, les images des deux caméras de droite et de gauche en corrigeant le label associé. Par exemple, une image de la caméra de gauche simule un cas où la voiture est déportée vers la gauche, on va donc apprendre au réseau de neurones à choisir un angle au volant positif pour ramener la voiture au centre de la voie (et symétriquement pour une image provenant de la caméra de droite, le réseau devra décider un angle au volant négatif). Nous parlons *d'augmentation de labels* car cette technique est très différente de la méthode *d'augmentation de données* utilisée régulièrement dans le domaine de la vision par ordinateur où les images sont légèrement modifiées pour diversifier les entrées mais où les labels restent inchangés. Cette approche permet en quelque sorte de collecter des données de cas d'échec sans avoir à les réaliser effectivement. Elle est illustrée en Figure 6.4.

C'est sur ce travail que se repose principalement notre article *End to End Vehicle Lateral Control Using a Single Fisheye Camera* [Toromanoff et al., 2018] publié à la conférence IROS 2018 à Madrid. De nombreux autres travaux [Yang et al., 2018, Hubschneider et al., 2017] se sont aussi inspirés de ce premier travail intégré sur voiture réelle pour eux aussi réussir à appliquer un apprentissage par imitation sur une voiture réelle pour du maintien de voie. Dans l'article de Hubschneider et al. [2017] par exemple, Hubschneider et al. [2017] ont rajouté à l'architecture initial de Nvidia une commande de navigation haut niveau en entrée du réseau de neurones. Cette commande provient des clignotants que le conducteur humain a utilisé en conduisant, i.e. aller à droite, aller à gauche ou aller tout droit. Ainsi, cela permet aux chercheurs d'indiquer quelle voie doit prendre l'agent lorsqu'il arrive à une intersection. Dans ce travail, ils utilisent aussi

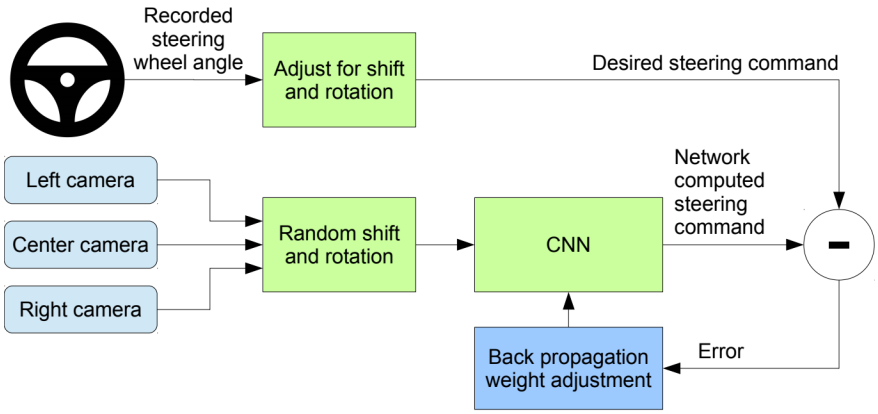


FIGURE 6.4 – Architecture de l'apprentissage par imitation introduite dans Bojarski et al. [2016]. L'idée principale est d'utiliser des caméras décalées à droite et à gauche de la caméra centrale afin de rajouter des situations d'échec dans la base de données d'entraînement. En effet, les cas d'échec pour le maintien de voie sont les situations où l'agent s'est déporté de sa voie. Rajouter des images issues des caméras décalées en adaptant l'angle au volant désiré pour ces images permet à l'agent d'apprendre à revenir au centre de sa voie lorsqu'il commence à dévier. Image issue de Bojarski et al. [2016].

plusieurs caméras frontales afin de générer des situations d'échec en suivant le même principe introduit dans l'article de Bojarski et al. [2016].

Il est important de noter que toutes les applications décrites précédemment apprennent uniquement le contrôle latéral pour du maintien de voie. En fait, il n'y a actuellement qu'un seul article publié d'imitation qui effectue du contrôle latéral et longitudinal sur un véhicule réel : *Urban Driving with Conditional Imitation Learning* [Hawke et al., 2020].

Dans cet article, Hawke et al. [2020] découplent l'apprentissage en trois parties distinctes, une partie perception, une partie fusion de capteurs (trois caméras) et finalement une partie contrôle. Ce découplage est illustré en Figure 6.5.

Cette approche ressemble en quelque sorte à notre approche des *indices implicites* [Toromanoff et al., 2020] introduit dans le chapitre précédent. Hawke et al. [2020] utilisent trois tâches de perception pour pré-entraîner leur encodeur : la prédiction de segmentation sémantique, la prédiction d'une carte de profondeur ainsi que la prédiction du flux optique, c.f. Figure 6.6 pour des exemples de ces tâches.

Ensuite Hawke et al. [2020] entraînent leur réseau par imitation sur plusieurs pas de temps, i.e. faire prédire au réseau les N prochaines actions (angles au volant et vitesses) prises par le conducteur humain. Prédire plusieurs pas de temps dans le futur permet d'obtenir des contrôles plus lisses selon Hawke et al. [2020]. Les chercheurs ont ensuite évalué de nombreux modèles différents en boucle fermée sur un véhicule réel. Leur métrique repose principalement sur le nombre de mètres parcourus en moyenne par l'agent avant une reprise en main. Ils ont évalué l'impact de ne pas pré-entraîner leur encodeur

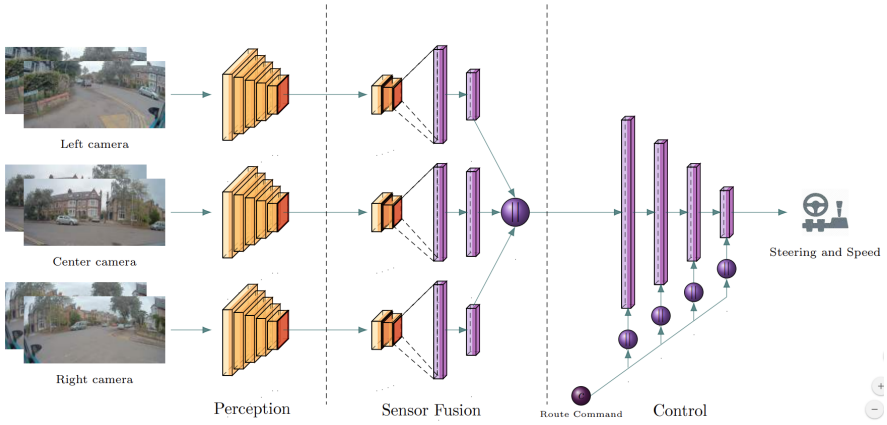


FIGURE 6.5 – Architecture de l'apprentissage par imitation utilisé par Hawke et al. [2020]. L'idée principale est de découpler l'apprentissage de la perception et l'apprentissage du contrôle. Ils vont donc d'abord entraîner leur encodeur sur certaines tâches haut niveau de perception et ensuite entraîner la suite du réseau à effectuer une fusion des informations provenant des différents capteurs afin de calculer le contrôle à effectuer. Image issue de Hawke et al. [2020].

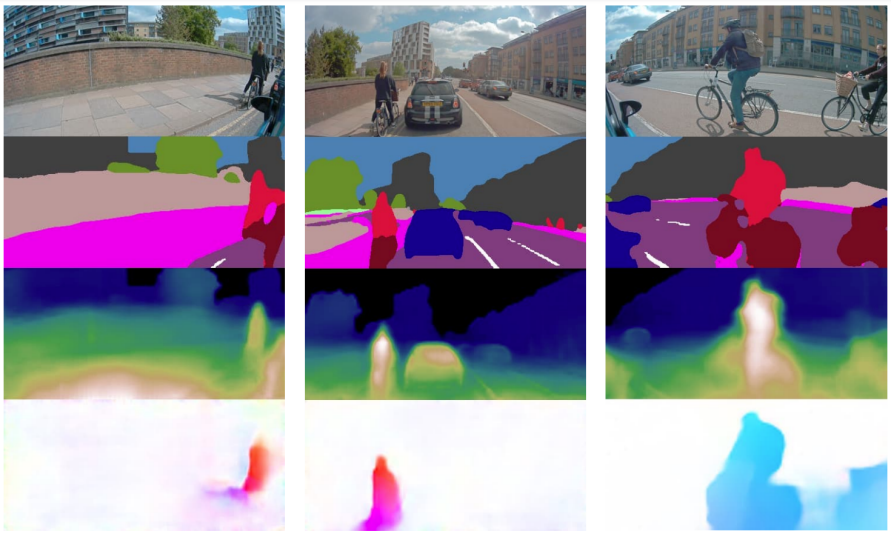


FIGURE 6.6 – Les trois tâches de perception pour pré-entraîner l'encodeur dans l'article de Hawke et al. [2020]. En haut l'image RGB d'entrée. Puis en dessous dans l'ordre, la prédiction de segmentation sémantique, la prédiction d'une carte de profondeur et finalement la prédiction du flux optique. Image issue de Hawke et al. [2020].

avec des tâches de perception, i.e. d'entraîner l'ensemble du réseau uniquement avec la fonction de coût d'imitation. Ils ont aussi testé d'utiliser une seule caméra frontale au lieu de trois. Finalement, ils ont étudié l'impact de réduire le nombre de données utilisé

lors de l'apprentissage. Ces résultats sont présentés sur la Table 6.1.

Modèle évalué	Taux d'intervention (m/int)
Une seule caméra (SV pour *Single View*)	222
3 caméras	306
3 caméras + flux	253
SV 75% données	154
SV 50% données	67
SV 25% données	44
SV sans perception	30
SV avec perception non fixée	177

TABLE 6.1 – Résultats des différents modèles évalués par Hawke et al. [2020]. Tout d'abord évaluation de l'influence du choix des entrées capteurs de l'agent. Dans un deuxième temps, l'impact de la taille de la base de données d'apprentissage. Finalement, l'impact de retirer le pré-entraînement sur des tâches de perception (*SV sans perception*) ainsi que d'entraîner l'imitation sans fixer les poids pré-appris de l'encodeur (*SV avec perception non fixée*). Résultats provenant de l'article de Hawke et al. [2020].

Le résultat le plus intéressant de ce tableau est l'impact de la phase de pré-entraînement par perception. De manière analogue à notre article sur CARLA [Toromanoff et al., 2020], les performances sont extrêmement faibles lorsque leur réseau est appris de bout-en-bout uniquement avec la fonction de coût d'imitation. De plus, il est très intéressant de noter qu'entraîner l'ensemble du réseau par imitation, même en ayant auparavant pré-appris l'encodeur avec les tâches de perception, amène des pertes de performances comparé à juste fixer les poids issus de ce pré-entraînement. Cela confirme encore plus l'intérêt de notre approche [Toromanoff et al., 2020] introduite dans le chapitre précédent pour effectuer de l'apprentissage par renforcement pour la voiture autonome. Finalement, comme prévu, les chercheurs ont trouvé que réduire la taille de la base de données d'apprentissage a un impact majeur sur les performances finales. De même, ajouter des caméras permet à l'agent de mieux percevoir le monde l'entourant et ainsi diminuer le nombre d'interventions. Par contre, rajouter le flux optique en entrée du réseau semble perturber l'apprentissage. Cela est probablement dû au problème de *confusion de causalité* qui peut survenir lors de la phase d'apprentissage par imitation car l'information de vitesse courante est comprise dans le flux optique. En effet, si on donne la vitesse courante à l'agent, il peut facilement juste prédire que la vitesse suivante soit égale à la vitesse courante et avoir un coût très faible mais ses performances en boucle fermée seront très mauvaises car l'agent conduira à une vitesse constante.

Cet article est donc très intéressant comme étant le premier à appliquer de l'apprentissage par imitation sur véhicule réel en milieu urbain pour le contrôle du volant et le contrôle de la vitesse. Cependant, les résultats finaux sont encore peu matures, leur meilleur agent ne roule que 300 mètres en moyenne avant une intervention. Cela est probablement dû, au moins en partie, au fait que Hawke et al. [2020] n'aient pas vraiment trouvé de solution pour gérer le problème de non-concordance de distributions. En effet, pour le moment les méthodes d'augmentation de labels ne sont appliquées que pour le

maintien de voie car il est relativement aisé de savoir quelles sont les situations d'échec dans ce cas d'application simple. Il est beaucoup moins évident de savoir quelles sont les situations d'échec et encore moins de réussir à les générer artificiellement pour le contrôle longitudinal. Ils restent donc de nombreuses améliorations à faire pour effectuer de la conduite urbaine sur voiture réelle avec de l'apprentissage par imitation.

6.1.2.2 Sur le simulateur CARLA

Comme nous l'avons mentionné dans le chapitre précédent, la grande majorité des approches évaluées sur le benchmark CARLA sont des approches utilisant de l'apprentissage par imitation. Ces articles doivent contrôler l'angle au volant ainsi que l'accélération et le freinage du véhicule. Contrairement aux approches indiquées précédemment sur voiture réelle, il est beaucoup plus aisé de générer des situations d'échec dans CARLA : par exemple il suffit de laisser l'agent conduire qui se retrouvera de lui même dans ces situations d'échec comme dans l'algorithme DAgger [Ross et al., 2011].

La toute première approche évaluée sur le benchmark CARLA est l'article de Codevilla et al. [2018]. Dans ce travail, les chercheurs se sont grandement appuyés sur l'article de Nvidia [Bojarski et al., 2016] et ont eux aussi utilisé trois caméras frontales pour générer des situations variées dans la base de données d'apprentissage. Ils ont aussi utilisé une augmentation de données et de labels tirant parti du fait que CARLA est un simulateur en rajoutant du bruit aux commandes de l'autopilote intégré dans le simulateur. De cette façon, ils ont pu générer de nombreux cas d'échec tout en ayant automatiquement la correction nécessaire pour revenir à une situation optimale. Finalement, ils ont introduit un réseau de neurones *conditionnel* permettant de gérer les intersections et donner une direction à suivre à l'agent. Ils ont en fait remarqué que juste donner une commande haut niveau en entrée de l'agent ne donnait pas des résultats optimaux. En effet, le comportement à appliquer pour aller à droite est très différent de celui à prendre pour aller à gauche ou tout droit, l'agent avait donc beaucoup de mal à apprendre cela juste avec une commande haut niveau en entrée. Ils ont donc utilisé un réseau de neurones avec plusieurs têtes différentes, une pour chaque commande de navigation. L'idée est que chaque tête va se spécialiser pour une commande de navigation précise. De plus, seulement une partie limitée du réseau est séparée en plusieurs têtes : l'impact sur le temps d'apprentissage est donc négligeable comparé à apprendre tout le reste du réseau qui est de toute façon partagé pour toutes les commandes haut niveau. Ce principe est illustré sur la Figure 6.7.

L'autre article d'imitation que nous allons détailler est l'approche de *Learning By Cheating* (LBC) [Chen et al., 2020a]. Comme nous l'avons vu précédemment, c'est en effet la seule approche qui a des performances équivalentes voire supérieures à la nôtre sur le benchmark CARLA et le benchmark *NoCrash*. Nous verrons que cette approche introduit une méthode permettant de générer des situations très variées qui permettent de réduire considérablement le problème de non-concordance de distributions. L'apprentissage est en fait séparé en deux phases, d'abord un agent privilégié apprend en trichant par imitation, i.e. apprend à imiter le contrôle de l'expert en ayant accès à des informations non-accessibles par l'agent final. Ces informations privilégiées sont une vue du

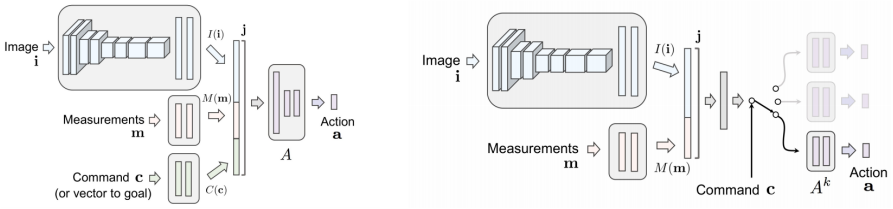


FIGURE 6.7 – Les 2 architectures conditionnées par une commande de navigation haut niveau présentées par Codevilla et al. [2018]. A gauche, l'architecture naïve consistant à donner en entrée du réseau un vecteur codant la commande de navigation. A droite l'architecture conditionnelle finalement utilisée. L'idée est de séparer la fin du réseau en plusieurs têtes, chaque tête ayant ses poids propres et se spécialisant pour une commande spécifique. Image issue de Codevilla et al. [2018].

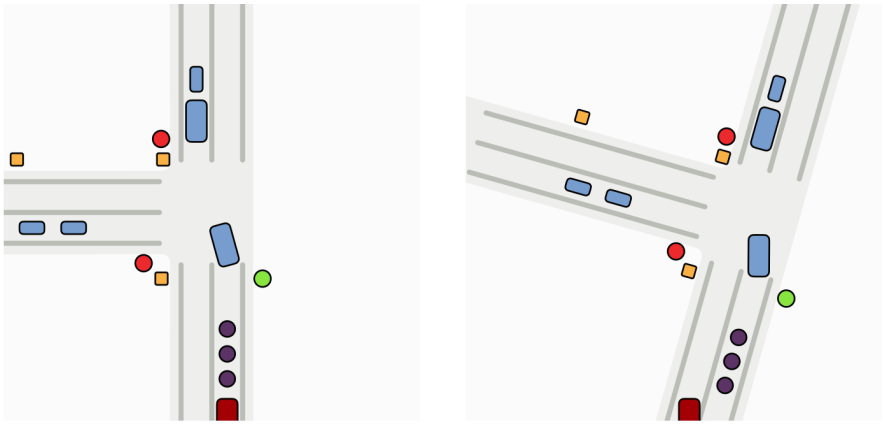
dessus où toutes les informations de perception comme l'état des feux tricolores, la position des voies ainsi que la position des véhicules et piétons environnants sont données. Une autre différence est que l'agent doit prédire une trajectoire et non plus des contrôles bruts. Cette représentation de l'état et des actions permet de très facilement générer des nouvelles situations pour pallier le problème de non-concordance de distributions. En effet, il suffit de faire une translation ou une rotation à l'image vue du dessus pour simuler un mauvais positionnement de l'agent, la trajectoire voulue est elle aussi soumise à la même transformation. l'intuition est que quelque soit le déplacement artificiellement généré, l'agent doit suivre la trajectoire que l'expert a suivie. La représentation vu du dessus ainsi que les augmentations de données et de labels sont illustrés Figure 6.8.

L'idée est ensuite d'entraîner un deuxième agent à imiter cet agent prilivigé. L'agent final lui n'a accès qu'aux informations brutes des capteurs et cherche à imiter l'agent privilégié en tant qu'expert. Cela est illustré sur la figure suivante, Figure 6.9.

Cette décomposition en deux phases d'apprentissage comporte plusieurs avantages. Tout d'abord le premier agent priviligié apprend à partir d'une représentation plus compacte de l'environnement ce qui permet d'apprendre bien plus vite et de mieux généraliser. De plus, cette représentation priviligiée permet de générer très facilement des situations d'échec et leurs corrections associées.

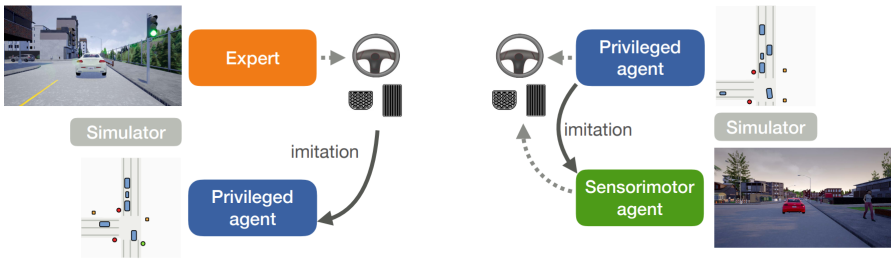
Deuxièmement, ce nouvel expert apporte un signal de supervision bien plus fort que l'expert initial. En effet, il peut être utilisé pour annoter automatiquement toutes les situations et non plus uniquement les situations qui ont été observées par l'expert initial. Cela permet d'effectuer automatiquement des algorithmes comme DAgger [Ross et al., 2011] permettant de rajouter des situations rencontrées lors de l'évaluation de l'agent avec leurs corrections associées.

Finalement, comme leur agent privilégié utilise un réseau conditionnel possédant quatre branches différentes pour les quatre commandes de navigation (tout droit, à gauche, à droite et rester sur sa voie), il peut générer automatiquement quatre conduites possibles pour chaque situation. Cela permet d'augmenter encore la variété dans la base d'entraînement et ainsi d'encore plus atténuer le problème de non-concordance de dis-



(a) Road map (b) Rotation and shift aug.

FIGURE 6.8 – La représentation privilégiée utilisée lors de la première phase d'apprentissage par Chen et al. [2020a]. L'agent (en rouge foncé) est situé en bas au centre de l'image et de nombreuses informations de perception sont indiquées comme la couleur des feux, la position des véhicules et des piétons environnants. La trajectoire à suivre que l'agent doit imiter est indiquée par les points violets. Cette représentation permet très facilement de générer des cas d'échecs, il suffit juste de faire une transformation à toute l'image ainsi qu'à la trajectoire à suivre comme présenté sur l'image de droite. Image issue de Chen et al. [2020a].



(a) Privileged agent imitates the expert (b) Sensorimotor agent imitates the privileged agent

FIGURE 6.9 – Les deux phases d'apprentissage utilisées dans l'article Learning By Cheating [Chen et al., 2020a]. D'abord un agent privilégié apprend à copier l'expert initial de CARLA. Cet agent est considéré comme tricheur car il utilise des informations qui ne sont pas accessibles dans l'évaluation finale. Dans un deuxième temps, un autre agent apprend à imiter ce premier agent. Ce second agent n'aura lui accès qu'aux données brutes des capteurs. Image issue de Chen et al. [2020a].

tributions.

En conclusion, cet algorithme introduit une méthode très intéressante qui réduit considérablement le problème fondamental de l'apprentissage par imitation et c'est pour cela qu'il est actuellement l'algorithme ayant les meilleurs performances. Cependant, il

est important de noter que leur méthode repose grandement sur le simulateur CARLA et qu'il ne serait pas possible d'appliquer leur méthode telle quelle sur une voiture réelle.

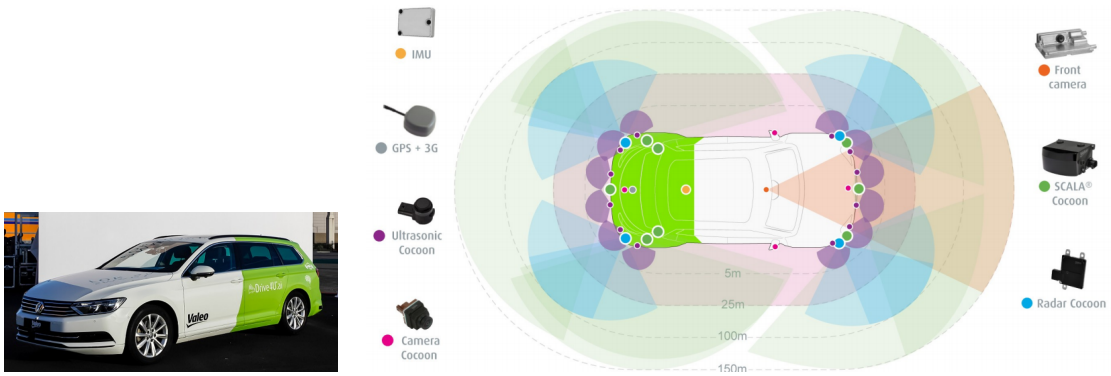
6.2 Contrôle latéral sur véhicule réel : configuration expérimentale

Après avoir présenté l'apprentissage par imitation ainsi que détaillé les travaux les plus représentatifs d'apprentissage par imitation pour la voiture autonome, nous allons maintenant présenter notre approche publiée à la conférence IROS 2018 [Toromanoff et al., 2018] intitulée : *End to End Vehicle Lateral Control Using a Single Fisheye Camera*. L'idée de notre travail était de ré-appliquer les travaux de Nvidia [Bojarski et al., 2016] sur les véhicules Valeo ne possédant qu'une seule caméra fisheye frontale. Nous nous intéressons donc uniquement au contrôle du volant pour une application de maintien de voie.

Dans cette section, nous présenterons d'abord les différents véhicules réels où nous avons intégré notre système, puis nous verrons quels ont été les cas d'utilisation ciblés, les base de données que l'on a utilisées pour l'apprentissage et finalement notre architecture de réseau de neurones.

6.2.1 Présentation des capteurs et des véhicules réels

Nous avons utilisé trois voitures différentes pour la collection de données et pour nos tests, cf Figure 6.10. Les trois véhicules sont équipés d'un système matériel similaire en particulier d'un bus CAN (pour *Controlled Area Network*) permettant d'obtenir la vitesse et l'angle au volant courants, ainsi qu'une caméra frontale fisheye.



(a) Une des voitures utilisées (b) Configuration des capteurs

FIGURE 6.10 – À gauche, une des trois voitures utilisées pour l'acquisition des données et l'intégration de l'apprentissage par imitation. À droite, la configuration des capteurs ainsi que leur champ de vision. Pour ce travail, nous avons seulement utilisé la caméra fisheye frontale (*Caméra Cocoon* situé à l'avant).

La caméra a un angle de vue horizontal de 190° et fournit une image RGB de résolution initiale de 1280x800 pixels à 30Hz. Un exemple d'une telle image fisheye est présentée en Figure 6.11. Sur chaque voiture, la caméra est située à une hauteur similaire, environ 50 centimètres au dessus du sol. Les deux premières voitures sont équipées avec un *drive-by-wire* actif qui permet de contrôler le volant via une MicroAutobox. Les inférences de notre réseau de neurones dans ces deux véhicules actifs sont effectuées sur une Nvidia DrivePX2 AutoChauffeur intégrée dans le véhicule. L'intégration des algorithmes dans le véhicule ainsi que l'architecture globale pour l'intégration ont été effectuées par d'autres ingénieurs du Driving Assistance Research à Valeo. Le troisième véhicule est passif et a été utilisé pour enregistrer la base de données initiale décrite dans la partie suivante. Il est en effet plus simple pour des raisons de sécurité d'enregistrer les données sur un véhicule passif.

FIGURE 6.11 – Exemple d'image fisheye haute résolution (1280x800). On peut distinguer des distorsions de l'image en particulier sur les cotés de l'image.

6.2.2 Les cas d'utilisation ciblés et les base de données associées

Dans cette section, nous allons décrire les deux cas d'utilisation ciblés. Le premier, la *route ouverte*, correspond à de la conduite en environnement réel, i.e. sur les routes avec d'autres véhicules, piétons et les panneaux et feux de signalisation standards. Le deuxième cas d'application, la *piste d'essai*, est un environnement fermé, il n'y a pas d'autres véhicules ou piétons et l'ensemble des obstacles sont contrôlés.

Route ouverte Pour entraîner sur des données de route ouverte, nous avons enregistré une base de données dans la région de Paris sous différentes conditions météorologiques. Cette base de données représente plus de 10 000 km et 200 heures de conduite sur route ouverte. Elle contient différents types de route : autoroute, environnement urbain, route de campagne etc (cf Figure 6.12). Nous avons séparé notre base de données en une base d'entraînement (environ 10 millions d'images) et une base d'évaluation (environ 3 millions). Toutes les images où le conducteur humain change de voie ou tourne à une

intersection sont retirées automatiquement lorsque l'un des clignotants est actionné. En effet, nous nous intéressons uniquement au cas où le conducteur humain reste sur sa voie. Nous avons aussi retiré toutes les images où la vitesse est très faible car dans ces situations l'angle au volant peut être extrêmement bruité : si le conducteur s'arrête avec les roues tournées, cela est impossible à apprendre pour le réseau de neurones et peut donc nuire à l'apprentissage.

(a) Ville (b) Route de campagne (c) Autoroute

FIGURE 6.12 – Projection cylindrique de l'image fisheye provenant de la caméra frontale dans des lieux représentatifs de notre base de données.

Piste d'essai présentée au CES 2018 à Las Vegas Nous avons aussi entraîné notre algorithme sur une piste d'essai dans l'optique d'une démonstration au CES (pour *Consumer Electric Show*) 2018 à Las Vegas. Comme illustré sur la Figure 6.13, la piste est composée de deux virages très serrés (16 mètres de diamètre), une barrière ouvrante, une partie toute droite ainsi qu'une déviation de la route principale avec une chicane de cônes de travaux. Les lignes sont masquées sur les deux virages en épingles afin de montrer qu'on peut gérer des situations où le contrôle classique fondé sur de la détection de lignes aurait des difficultés.

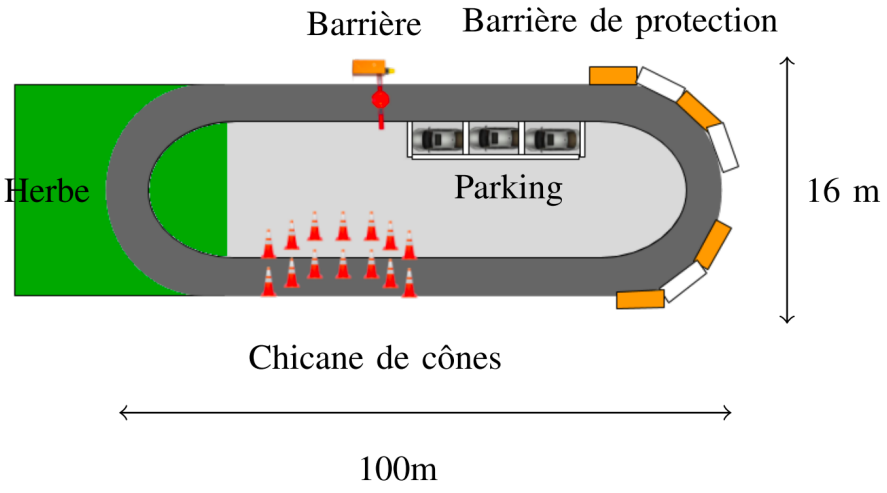


FIGURE 6.13 – Illustration de la piste d'essai présentée au CES 2018 à Las Vegas. La piste possède deux virages en épingle, une barrière, une chicane de cônes ainsi qu'une partie sans ligne avec des voitures garées sur le coté.

Le but pour cette démonstration est donc d'adapter l'angle au volant du véhicule : d'abord rester sur sa voie sur la section droite, puis passer les deux virages en épingle (dans lesquels l'angle au volant peut aller jusqu'à plus de 500°) pour finalement se déporter dans la chicane de cônes sans en heurter. Le contrôle longitudinal, c'est à dire l'accélération et le freinage, est soit exécuté manuellement par un conducteur humain, soit contrôlé par un autre réseau de neurones (c'était le cas pour la démonstration en direct au CES, voir l'article de George et al. [2018]).

La piste d'essai contient des scénarios qui ne sont pas présents lors de la conduite en route ouverte. C'est pour cela que nous avons enregistré une base de données spécialement sur la piste d'essai pour pouvoir gérer ces situations inhabituelles. Un autre point important de ces enregistrements spécifiques était de rassembler le plus de variabilité possible en terme de luminosité et de conditions météorologiques afin d'obtenir un comportement final plus robuste.

Nous avons enregistré cette base de données spécifique de la manière suivante : d'abord 10 tours de piste sont enregistrés pour la base d'entraînement suivis de 3 tours de piste gardés pour la validation et l'évaluation. Les sessions d'enregistrements étaient étalées tout le long de la journée et séparées chacune d'environ une heure. Cela a permis d'avoir des données du lever du soleil jusqu'au coucher. La base de données finale a été enregistrée sur 5 jours consécutifs avec environ 8 sessions d'enregistrements par jour, résultant en une base de données d'environ 360 000 images d'entraînement et 110 000 images gardées pour la validation.

6.2.3 Architecture de notre réseau de neurones

Notre architecture est très proche de l'architecture introduite par Bojarski et al. [2016]. Notre réseau comporte dix couches et prend en entrée une image de taille 200x66. Il est décrit précisément sur la Table 6.2.

Type	Taille	Nombre de filtres	Stride
Normalisation	200x66x3	N/A	N/A
Conv #1	5x5	24	2
Conv #2	5x5	36	2
Conv #3	5x5	48	2
Conv #4	3x3	64	1
Conv #5	3x3	64	1
FC #1	1152x1164	N/A	N/A
FC #2	1164x100	N/A	N/A
FC #3	100x50	N/A	N/A
FC #4	50x10	N/A	N/A
FC #5	10x1	N/A	N/A

TABLE 6.2 – Architecture de notre réseau de neurones avec le type de couche, les parmètres et la taille (Conv pour *Convolution* et FC pour *Fully Connected*).

Notre réseau a été entraîné à minimiser la fonction de coût quadratique entre l'angle au volant calculé par le réseau et la vérité terrain, c'est à dire l'angle au volant pris par le conducteur humain. Nous avons aussi ajouté une régularisation L2 pour éviter le sur-apprentissage. La fonction de coût finale utilisée est définie ci-dessous (Équation 6.1) avec θ les paramètres de notre réseau et λ_{reg} le poids associé à la régularisation :

$$L(Y, \hat{Y}, \theta) = L_{MSE}(Y, \hat{Y}) + \lambda_{reg} L2_{reg}(\theta) \tag{6.1}$$

6.3 Méthode et contributions

Dans cette section nous exposerons notre méthode et nos contributions. Nous allons d'abord expliciter notre contribution principale : gérer le problème de non-concordance de distributions en utilisant uniquement une seule caméra frontale contrairement à Bojarski et al. [2016] ou à Hubschneider et al. [2017]. Puis nous détaillerons notre deuxième contribution : un simulateur fondé sur des images réelles et un modèle physique réaliste du véhicule permettant d'évaluer le comportement de notre réseau de neurones en boucle fermée. Finalement, nous présenterons les problèmes de répartition de données présents dans notre base de données initiale et les différents choix de sélection de données que l'on a effectués.

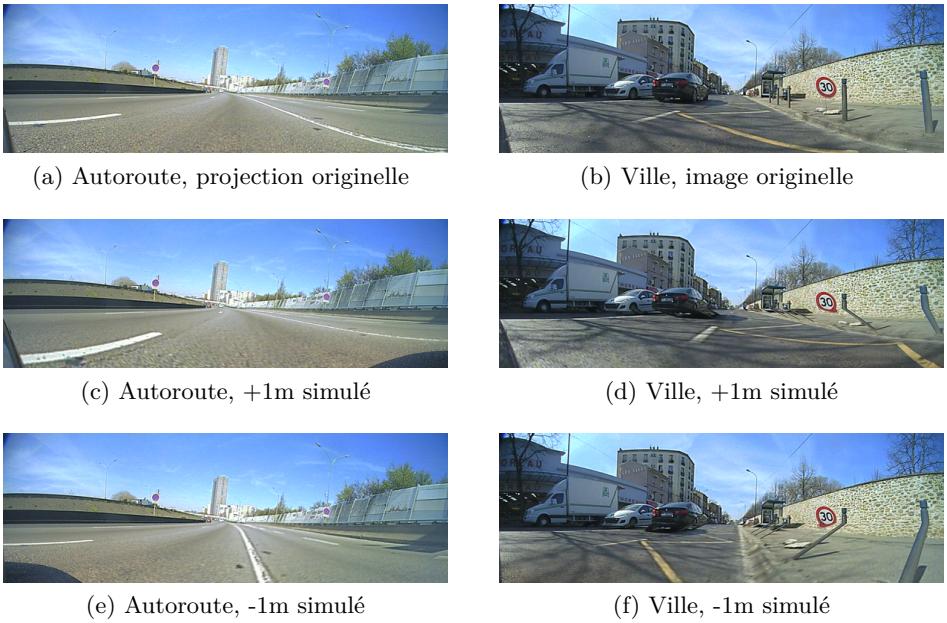
6.3.1 Augmentation de point de vue et correction de l'angle au volant associé avec une seule caméra fisheye

Pour gérer le problème fondamental de non-concordance de distributions, nous nous sommes inspirés de l'augmentation de labels introduite dans l'article de Bojarski et al. [2016]. Le but de l'augmentation de labels est de d'abord simuler les images qui seraient obtenues si le véhicule était légèrement déporté de la position originelle du conducteur humain, puis calculer l'angle au volant qu'il faudrait effectuer pour revenir à la position idéale et finalement rajouter cette image simulée et son angle au volant associé à la base de données d'apprentissage. Contrairement à Bojarski et al. [2016] qui utilisent trois caméras pour générer leur transformation d'image, notre contribution principale est d'utiliser uniquement une seule caméra frontale pour simuler différents points de vue. Cela est possible grâce au champ de vision latéral de la caméra fisheye qui est bien supérieur au champ de vision d'une caméra standard. De plus, cela permet de réduire les contraintes d'intégration. En effet, nous avons ainsi besoin que d'une seule caméra frontale pour l'évaluation ainsi que pour la phase d'entraînement : cela a en particulier permis d'utiliser la base de données de 10 000 km présentée précédemment (Section 6.2.2) qui ne comporte qu'une seule caméra frontale.

Simulation de point de vue Pour générer nos augmentations de point de vue, nous avons tout d'abord effectué une projection cylindrique sur un plan perpendiculaire au sol. Cela permet de limiter les distorsions induites par le large champ de vision. De

plus, cette projection est invariante selon l'orientation de la caméra sur le véhicule et permet d'avoir des images homogènes entre nos trois voitures, en prenant en compte que la caméra est située à la même hauteur sur les trois véhicules. Finalement, l'image est tronquée et redimensionnée à une résolution plus faible avec une interpolation linéaire. Des exemples avec différentes sections d'images et différentes résolutions sont présents sur la Figure 6.17.

Avec cette projection cylindrique, simuler des rotations du véhicule est très simple : il suffit de décaler l'image sur le coté, le décalage étant proportionnel à l'angle voulu. Par contre, il nous faut la position dans le monde (en 3D) des pixels pour simuler des translations du véhicule. En pratique, comme la calibration de la caméra est connue, il suffit de connaître une seule coordonnée monde des pixels pour connaître la position 3D du pixel dans le monde. En effet, les coordonnées du pixel dans l'image nous donne les caractéristiques d'un rayon, ainsi nous n'avons besoin que d'une seule coordonnée sur ce rayon pour déterminer la position exacte du point dans le monde. Cette coordonnée dans l'espace monde n'étant pas disponible, nous avons donc fait l'hypothèse que les points en dessous de l'horizon sont sur le sol (z=0) et que tous les points au-dessus sont à l'infini, nous appellerons cette hypothèse *sol/ciel* pour le reste de ce manuscrit. Cela permet d'abord de projeter chaque pixel dans l'espace monde (avec trois coordonnées) puis d'appliquer une translation sur ces coordonnées pour finalement les reprojeter dans l'espace image.



(a) Autoroute, projection originelle

(b) Ville, image originelle

(c) Autoroute, +1m simulé

(d) Ville, +1m simulé

(e) Autoroute, -1m simulé

(f) Ville, -1m simulé

FIGURE 6.14 – Exemple d'augmentations de point de vue latérale (sur autoroute et route en ville). A gauche, la déformation induite par notre reprojection est à peine visible. A droite par contre, on voit que les objets verticaux sont déformés en-dessous de l'horizon.

Cependant cette transformation génère des artefacts lorsque notre hypothèse n'est pas vérifiée, comme sur les objets en 3D comme les autres véhicules, mais pas sur la partie de l'image pertinente pour le contrôle latéral : la route, les marquages au sol ainsi que la perspective au loin. La Figure 6.14 illustre l'impact de différentes translations sur l'image générée : sur les images d'autoroute, la déformation est à peine visible, par contre sur les images en ville, on voit que les objets verticaux sont déformés en-dessous de l'horizon. Globalement, l'apparence est plus réaliste que la déformation d'image introduite par Hubschneider et al. [2017]. Nous soutenons que ces déformations ont un impact limité sur les performances finales comparées à l'amélioration apportée par cette augmentation de point de vue.

Génération de l'angle au volant associé Pour générer la nouvelle vérité terrain, i.e. l'angle au volant qu'il faut prendre pour revenir à une position idéale, nous utilisons un contrôleur latéral inspiré de Snider et al. [2009] et de Hoffmann et al. [2007] pour modéliser la trajectoire du conducteur humain. Nous faisons l'hypothèse que le conducteur suit une trajectoire en effectuant l'action suivante sur l'angle au volant :

$$\delta_h(t) = f(\kappa(t), v) + K_e(v)e(t) + K_\theta(v)\theta(t) \tag{6.2}$$

avec $\delta_h(t)$ l'angle au volant du conducteur, $f(\kappa(t), v)$ une fonction de la courbure de la route $\kappa(t)$ et de la vitesse du véhicule v, $e(t)$ et $\theta(t)$ sont respectivement l'erreur latérale et angulaire que le conducteur cherche à minimiser. Finalement, $K_e(v)$ et $K_\theta(v)$ sont des gains de contrôle pour l'erreur latérale et l'erreur angulaire. Leurs valeurs sont définies en utilisant des contrôleurs déjà développés sur les véhicules. Pour le cas des trois véhicules utilisés, on a $K_e(v) = 12/v\ m^{-1}$ et $K_\theta(v) = 5.3$.

Ainsi, pour obtenir l'angle au volant associé à une position générée $e(t) + \Delta e$ et à une rotation $\theta(t) + \Delta\theta$, il suffit de rajouter $K_e(v)\Delta e + K_\theta(v)\Delta\theta$ à l'angle au volant $\delta_h(t)$ initial du conducteur humain. Cela permet d'obtenir l'angle au volant augmenté $\delta_{augm}(t)$ directement depuis l'angle au volant initial sans avoir à calculer les variables inconnues $f(\kappa(t), v)$, $e(t)$, $\theta(t)$

$$\delta_{augm}(t) = \delta_h(t) + K_e(v)\Delta e + K_\theta(v)\Delta\theta \tag{6.3}$$

6.3.2 Simulateur fondé sur des images réelles

Comme indiqué précédemment, et corroboré dans certains articles comme Bansal et al. [2018] ou Bojarski et al. [2016], nous sommes convaincus que l'augmentation de labels est essentielle pour faire fonctionner l'apprentissage par imitation et que l'erreur moyenne n'est pas une métrique valide pour comparer les performances car cette métrique ne prend pas en compte l'accumulation d'erreurs. Un réseau appris avec des exemples augmentés pourrait être moins performant (en erreur moyenne) que le même réseau entraîné sans exemples augmentés, pourtant le deuxième échouera totalement à

conduire une voiture réelle. Cela montre que pour mesurer réellement les performances de notre réseau de neurones et évaluer l'impact de l'augmentation de labels, nous devons nécessairement faire une évaluation en boucle fermée, i.e l'action prise par le réseau doit influer l'image suivante afin d'évaluer si l'agent réagit correctement à ses propres erreurs.

La métrique idéale serait de laisser le réseau conduire sur la voiture réelle sur différents scénarios et d'enregistrer le temps moyen où le réseau conduit sans nécessité d'intervention humaine, mais cela est dangereux et coûteux en pratique. C'est pour cette raison que nous avons construit un simulateur fondé sur des images réelles, inspiré du simulateur introduit dans l'article initial de Bojarski et al. [2016]. Il est important de noter qu'au moment de la publication de notre article, l'ensemble des articles d'imitation pour le contrôle latéral (sauf l'article initial de Nvidia [Bojarski et al., 2016]) n'utilisaient pas ce type de simulateur et rapportaient leur performance uniquement en boucle ouverte. C'est pour cela que nous n'avons pas pu nous comparer à l'ensemble de ces travaux [Hubschneider et al., 2017], [Yang et al., 2018].

Notre simulateur repose sur la base de données et l'augmentation de labels présentées précédemment. L'idée est de comparer l'angle calculé par notre réseau de neurones à celui pris par le conducteur humain. En utilisant un modèle physique du véhicule (nous avons utilisé un *modèle bicyclette* [Kong et al., 2015, Snider et al., 2009]) nous pouvons estimer la position de la voiture réelle et estimer la position où aurait été la voiture si le réseau avait conduit. Nous avons donc utilisé deux modèles bicyclette en parallèle, un pour estimer la position réelle prenant en entrée les actions prises par le conducteur humain, et un pour estimer la position si le réseau de neurones avait conduit. A chaque pas de temps, nous obtenons ainsi un écart entre ces deux positions, cet écart est utilisé pour simuler l'image en entrée de notre agent. Le schéma du principe de notre simulateur est présenté en Figure 6.15.

Ce simulateur permet d'obtenir les performances en boucle fermée lorsque notre réseau de neurones conduit. Lorsque le réseau s'éloigne de plus d'un mètre en latéral de la vraie trajectoire enregistrée, nous admettons que cela aurait impliqué une intervention humaine car le réseau de neurones s'est trop déporté de sa voie. On peut donc utiliser la métrique idéale explicitée précédemment, le nombre d'interventions pour une certaine distance parcourue, grâce à notre simulateur. En pratique, ce simulateur était assez précis pour permettre d'obtenir une estimation fiable du comportement du réseau testé dans la voiture réelle.

La Figure 6.16 présente une capture d'écran de notre simulateur : la vraie trajectoire et la trajectoire du réseau sont représentées ainsi que l'image générée avec l'écart actuel entre le conducteur et le réseau.

6.3.3 Sélection de données

Dans notre base de données initiale, plus de 90% des angles au volant sont compris dans le segment [-10, 10] degrés (correspondant à [-0.5, 0.5] degré pour l'angle aux roues). C'est pour cela qu'un réseau entraîné sur la totalité de la base de données est biaisé vers les angles très faibles et va globalement toujours aller tout droit. Pour éviter ce biais, nous avons évalué trois sélections de données différentes. Pour notre premier essai, *Sélection*

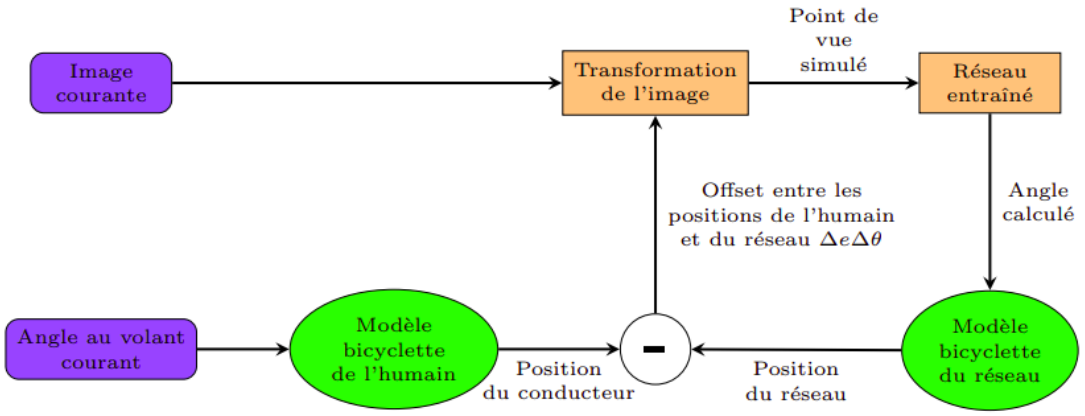


FIGURE 6.15 – Architecture de notre simulateur fondé sur images réelles. Les images enregistrées sont transformées en fonction de l'écart en position de la trajectoire enregistrée du conducteur humain et celle en boucle fermée du réseau de neurones. Pour connaître cet écart entre les deux trajectoires, nous maintenons deux modèles bicyclette en parallèle. Le premier prend en entrée les actions prises par le conducteur humain pour estimer la position de la voiture réelle, le deuxième prend en entrée les angles au volant calculés par notre réseau de neurone et permet d'estimer la position de la voiture si notre agent avait conduit.

FIGURE 6.16 – Capture d'écran de notre simulateur, la trajectoire bleue est celle du conducteur humain, en rouge la trajectoire du véhicule conduit par le réseau de neurones. A l'instant capturé, le réseau est décalé de 7cm à droite par rapport à la trajectoire originelle. On peut voir sur la trajectoire rouge que le réseau n'a pas réussi à prendre le virage précédent, il est sortie de sa voie. Une reprise en main a donc été effectuée, i.e. nous avons remis le réseau à la position de la trajectoire du conducteur humain.

1, nous avons retiré aléatoirement environ 50% des angles faibles de la base de données d'apprentissage. Pour la *Sélection 2*, nous avons retiré cette fois environ 85% des angles faibles. Le but est de voir comment il faut équilibrer notre base de données afin d'avoir les meilleures performances possibles à la fois sur les sections de route droite et les virages.

Finalement, nous avons aussi essayé de sur-représenter les données avec des angles au volant très élevés, *Sélection sur-représentée* pour voir si cela améliore les performances dans les virages. Ces distributions peuvent être caractérisées par leur déviation standard (std), voir Table 6.3.

Sélection	Std angle volant	Nombre angles faibles
Originelle	21°	5M
Sélection 1	26.4 °	2.6M
Sélection 2	35.3 °	0.9M
Sur-représentée	56 °	0.9M

TABLE 6.3 – Caractéristiques des différentes distributions d'angle au volant évaluées : la déviation standard (std) des angles au volant ainsi que le nombre de données dans l'intervalle [-10°, 10°].

Nous avons aussi évalué l'impact du champ de vision et de la résolution de l'image d'entrée pour savoir quelle projection utilisée. Les résultats de toutes ces expériences sont présentés dans la section suivante, Section 6.4.

6.4 Expériences

Dans cette section, nous présenterons les résultats de nos différents modèles en utilisant notre simulateur. Nous détaillerons d'abord la métrique que nous avons utilisée ainsi que les résultats quantitatifs de nos différents réseaux entraînés. Nous montrerons ensuite des résultats qualitatifs sur des environnements jamais vus, d'abord dans un autre simulateur, puis sur véhicule réel en France et aux États-Unis.

6.4.1 Définition de scénarios d'évaluation pour le simulateur

Pour comparer nos différents entraînements et avoir une idée précise du comportement du véhicule conduit par notre agent, nous avons construit une base de données d'évaluation représentative. Nous avons choisi manuellement des séquences variées de la base de données initiale de test. Finalement, les séquences d'évaluations sont séparées en trois scénarios principaux : urbain, autoroute (et route de campagne) et virage serrés (angle au volant $> 100°$). Ces scénarios sont décrits sur la Table 6.4.

Scénario	Urbain	Autoroute	Virages serrés
Nombre d'images	100000	70000	15000
Durée (min)	56	39	8

TABLE 6.4 – Description des différents scénarios d'évaluation dans notre simulateur.

Pour comparer quantitativement nos différents agents, nous les avons tous évalués dans le simulateur. Tout d'abord, nous calculons le nombre de reprises en main dans

chacun des trois scénarios. Lorsque le véhicule conduit par l'agent est trop loin de la trajectoire du conducteur, nous admettons qu'il y a eu une reprise en main et nous remettons l'agent sur la position du conducteur humain : nous avons choisi un mètre comme distance maximale avec la trajectoire humaine comme dans Bojarski et al. [2016]. Nous pouvons ensuite en déduire le pourcentage d'autonomie en utilisant la métrique introduite par Nvidia [Bojarski et al., 2016] pour chaque scénario et chaque agent entraîné. Cette autonomie est définie par l'équation suivante (Équation 6.4) avec R le nombre de reprises en main, t_r le temps pour revenir à une position idéale (on a pris 6s comme dans Bojarski et al. [2016]) et T le temps total pendant lequel l'agent conduit.

$$autonomie = 1 - \frac{R\ t_r}{T} \tag{6.4}$$

La moyenne de la distance (en valeur absolue) en translation entre l'humain et le réseau est aussi calculée. Cependant cette moyenne est biaisée par le fait qu'on remet le réseau à la position de l'humain après chaque reprise en main. En pratique, l'autonomie est donc notre métrique principale, la moyenne de la distance est utilisée pour départager les agents ayant un nombre de reprises en main très proche.

6.4.2 Résultats des différentes sélections de données

Sur la Table 6.5, nos agents entraînés avec différentes sélections de la base de données sont comparés avec une référence qui va toujours tout droit, i.e applique un angle de 0° quelle que soit l'entrée. Comme nous n'utilisons qu'une seule caméra fisheye frontale, les autres approches de l'état de l'art ne peuvent pas être comparées car leur méthode d'augmentation de labels ne peut pas être répliquée. Nous pouvons voir de ce tableau que la distribution initiale permet d'obtenir de bonnes performances en environnement urbain et sur autoroute mais comme prévu les performances en virages serrés ne sont pas satisfaisantes. De plus, enlever une proportion des angles faibles (*sélection 1 et 2*) améliore les performances sur les virages serrés tout en gardant des performances comparables en urbain et sur autoroute.

Scénario	Urbain		Autoroute		Virages serrés	
Métrique	Aut. (%)	MAD (cm)	Aut. (%)	MAD (cm)	Aut. (%)	MAD (cm)
Originelle	99.3	16	**98.7**	**19**	**73.7**	**30**
Sel. #1	98.9	**15**	97.7	25	83.7	**27**
Sel. #2	**99.5**	16	97.2	24	**87.5**	28
Sur-représenté.	**98**	**18**	**91.8**	**29**	82.5	29
Référence	8	36	14	41	0	35

TABLE 6.5 – Autonomie (%) et moyenne absolue de la distance (MAD, en cm) sur chaque scénario en fonction de la base de données d'apprentissage utilisée, la référence va toujours tout droit.

Par contre, la *sélection sur-représentée* a des performances moins bonnes que toutes les autres en urbain et sur autoroute. Et même sur les virages serrés, les résultats sont moins bons que les deux autres sélections de données. Nous supposons que cette distribution est trop éloignée de la distribution initiale (voir Table 6.3).

Les évaluations avec différents champs de vision et sections de l'image (Figure 6.17) ont montré qu'utiliser un champ de vision plus grand donnait des performances moindres, environ deux fois plus de reprises en main pour chaque scénario. Nous supposons que bien que le réseau pourrait en théorie apprendre à ignorer une partie de l'image d'entrée, on peut obtenir un meilleur apprentissage si on donne en entrée du réseau seulement l'information pertinente, i.e la route devant le véhicule. Cette expérience a confirmé notre choix initial de couper le ciel.

(a) 90 degré d'angle de vue (b) 135 degré d'angle de vue avec une image moins coupée

FIGURE 6.17 – Exemple de résolution et d'angle de vue différent.

6.4.3 Bagging de différents modèles

Scénario	Urbain		Autoroute		Virages serrés	
Métrique	Aut. (%)	MAD (cm)	Aut. (%)	MAD (cm)	Aut. (%)	MAD (cm)
Agent #1	**99.5**	16	97.2	24	**87.5**	28
Agent #2	98.9	15	97.7	25	83.7	**27**
Agent #3	99.3	16	**98.7**	**19**	**73.7**	**30**
Agent #4	98.6	**18**	**92**	**26**	85	29
Agent #5	**98.4**	15	96.4	21	83.7	28
Bagging	**99.5**	**13**	**98.7**	**19**	**87.5**	**27**

TABLE 6.6 – Comparaison des performances entre chaque réseau individuel et leur bagging, i.e. effectuer l'angle au volant correspondant à la moyenne des sorties des 5 agents.

Prendre les sorties de plusieurs agents et les moyenner peut améliorer les performances finales, même si ici nos sorties sont continues contrairement à notre agent appris par renforcement présenté dans le chapitre précédent. Cela est une méthode d'apprentissage d'ensemble (*ensemble learning* en anglais) appelé *Bagging*. Nous avons pu mesurer

quantitativement cette amélioration dans notre simulateur et avons remarqué que le gain était plus grand lorsque l'on moyennait les sorties d'agents appris avec différentes sélections de données. Cela diffère du Bagging standard car les différentes bases de données d'apprentissage sont issues de distributions différentes de la base de données initiale. Table 6.6 montre un exemple d'un tel gain en performance via ce Bagging. Tous les agents sont entraînés sur des bases de données différentes : différentes sélections de vitesse et de distributions d'angle au volant. Nous pouvons voir que sur chaque scénario, la performance finale du Bagging est meilleure que celle de tous les agents individuels.

6.4.4 Résultat qualitatif sur Grand Theft Auto (GTA)

Un résultat qualitatif intermédiaire très intéressant fut de tester notre agent dans le jeu vidéo GTA. Le réseau était capable de conduire alors qu'il a été entraîné seulement sur des images réelles et n'avait jamais vu d'image issues de GTA. De plus, comme indiqué dans le Chapitre 2 (section 2.5.1), GTA est un simulateur avec une physique et des graphismes très réalistes ce qui est prometteur quant à la capacité de généralisation de notre agent appris par imitation. La vidéo est disponible ici : https://youtu.be/y8yJQ0jGnco. Ce résultat prouve la capacité de généralisation de notre agent et nous a amené aux tests sur voiture réelle.

6.4.5 Résultats sur voiture réelle

Une fois que nous avons obtenus des résultats satisfaisants sur notre simulateur, nous avons intégré notre réseau sur une voiture réelle, pour montrer qu'il était capable de conduire de manière robuste sur un véhicule réel. Une vidéo qualitative de la performance de notre réseau peut se trouver ici https://youtu.be/arBrxGDXBxQ (la vitesse est contrôlée par le conducteur humain). Nous avons remarqué que notre réseau était robuste à différentes conditions de luminosité et météo. Nous avons aussi testé notre réseau sur un autre véhicule situé aux États-Unis avec des résultats concluants alors que la base d'entraînement n'avait été enregistré qu'en France. Finalement, nous avons aussi démontré la performance de notre approche au CES 2018 à Las Vegas (la vidéo est disponible ici https://youtu.be/wqXR71qVZk4). A noter que pour cette démonstration, la vitesse est contrôlée par un autre réseau de neurones appris par imitation.

6.5 Conclusion

Nous avons présenté un apprentissage par imitation de bout-en-bout intégré dans un véhicule réel pour le contrôle de l'angle au volant prenant une entrée une image d'une caméra fisheye. En tirant parti du champ de vision très large de la caméra fisheye, nous avons construit une nouvelle technique pour générer des points du vue réalistes qui simulent des cas d'échec pour le contrôle latéral. Ces images générées sont ensuite associées avec un angle au volant corrigé utilisant des modèles de contrôle pré-existants sur nos véhicules. De cette manière, les situations d'échec sont générées avec une seule caméra

frontale contrairement aux travaux de l'état de l'art [Bojarski et al., 2016, Hubschneider et al., 2017, Yang et al., 2018] qui utilisent plusieurs caméras frontales pour cela.

Cette génération de point de vue est ensuite combinée avec un modèle physique du véhicule afin de construire un simulateur fondé sur des images réelles, qui est ensuite utilisé pour évaluer le comportement en boucle fermée de nos agents. Cela est totalement différent des approches en boucle ouverte où l'accumulation d'erreurs du réseau n'est pas prise en compte. Nous utilisons notre simulateur pour déterminer un pourcentage d'autonomie qui dépend du nombre de reprises en main et de la distance parcourue. Nous avons ainsi pu montrer que dans notre simulateur réaliste, notre meilleur agent avait une autonomie de plus de 99% en environnement urbain. Nous avons aussi validé notre approche qualitativement sur plusieurs véhicules réels, d'abord en route ouverte et ensuite sur une piste d'essai comportant des cas d'usages difficiles comme des virages en épingles et des zones de travaux.

Cette première expérience sur voiture réelle nous a permis de bien appréhender les différents capteurs et les bases de données réelles. Cela nous a aussi permis d'intégrer notre travail dans une vraie voiture et de faire face aux nombreuses difficultés qui surviennent lors de l'intégration. Dans le dernier chapitre de cette thèse, Chapitre 7, nous verrons comment appliquer véritablement de l'apprentissage par renforcement sur des données réelles. Ces derniers travaux se sont reposés en grande partie sur cette première intégration sur véhicule réel, en particulier sur la base de données initiale de 10 000km et surtout sur le simulateur fondé sur des images réelles.

Chapitre 7

Apprentissage par renforcement sur un véhicule réel

7.1 Introduction

Dans le chapitre précédent, nous avons intégré un apprentissage par imitation pour le contrôle latéral sur un véhicule réel. Nous avons en particulier décrit comment nous avons évalué nos différents agents appris par imitation en utilisant un simulateur fondé sur des images réelles. Dans le dernier chapitre de cette thèse, nous présenterons notre travail consistant à appliquer du renforcement sur des données réelles. Le but final de ce chapitre est de réussir à faire fonctionner de l'apprentissage par renforcement pour du contrôle latéral et longitudinal dans un environnement urbain. L'intégration sur un véhicule réel n'a malheureusement pas pu être effectuée dans le temps imparti de cette thèse, en particulier à cause de la situation sanitaire liée au Covid-19. Cette intégration sera poursuivie dans des travaux futurs.

Dans ce chapitre, nous détaillerons d'abord les travaux appliquant de l'apprentissage par renforcement sur un véhicule réel. Il y a en fait seulement deux travaux publiés dans ce cadre au moment de la rédaction de cette thèse, *Learning To Drive in a Day* [Kendall et al., 2019] et VISTA [Amini et al., 2020]. Comme nous le verrons dans la section suivante, ces deux travaux s'appliquent seulement au contrôle latéral d'une voiture réelle pour du maintien de voie. En fait, le deuxième article VISTA [Amini et al., 2020], apporte des idées extrêmement intéressantes qui ont fortement inspiré l'ensemble des travaux de ce chapitre.

Dans un second temps, nous détaillerons notre simulateur fondé sur des données réelles utilisant une estimation de profondeur à partir d'une seule caméra. Nous exposerons ensuite notre approche et nos résultats pour le contrôle latéral et longitudinal en environnement urbain. Pour résumer, notre approche générale est de combiner nos travaux effectués sur le simulateur CARLA (i.e. les *indices implicites*) avec les idées introduites par Amini et al. [2020]. Pour le contrôle latéral, notre approche s'inspire grandement de l'article VISTA mais s'applique à un cas plus général pour gérer les décisions aux intersections avec l'ajout d'une commande de navigation haut niveau. Finalement, nous exposerons nos premiers résultats pour le contrôle longitudinal en conduite urbaine. Ces derniers travaux sont toujours en développement au moment de l'écriture de ce manuscrit et nos résultats pour le contrôle longitudinal ne sont pas encore assez matures. Cependant, il est important de noter qu'à ce jour, il n'y a aucun travail publié de renforcement appliqué au contrôle longitudinal sur un véhicule réel et que nos résultats pour le contrôle longitudinal sont tout de même encourageants.

7.2 Travaux connexes : DRL appliqué sur un véhicule réel

7.2.1 Learning to Drive in a Day : Première application sur véhicule réel

Peu de temps après le début de cette thèse est sorti le premier article [Kendall et al., 2019] appliquant du renforcement sur une vraie voiture pour du maintien de voie. Ici, seul le contrôle latéral est appris, le contrôle longitudinal est laissé à un conducteur humain. Ce résultat est particulièrement impressionnant en particulier au niveau de l'intégration. C'est en effet un algorithme d'apprentissage par renforcement profond qui apprend directement sur une voiture réelle à partir de zéro.

Pour cela, Kendall et al. [2019] ont d'abord cherché en simulation des hyperparamètres et une architecture de réseau de neurones capable d'apprendre avec le moins d'itérations possibles afin de réussir à apprendre directement dans le monde réel avec très peu de données. Ils ont aussi utilisé un *Variational AutoEncoder* [Kingma and Welling, 2013] pour compresser l'état (l'image de la caméra frontale) et ainsi accélérer encore l'apprentissage.

Cependant, ce travail bien que très novateur est en fait trop simple pour pouvoir être appliqué à un cas d'usage plus complexe comme la conduite en environnement urbain. En effet, le cas d'application est du maintien de voie sur une portion de route de 250 mètres, l'algorithme n'a donc qu'assez peu de capacité de généralisation à d'autres types de route. De plus, ils ont appris directement dans le véhicule réel en utilisant les reprises en main du conducteur humain comme seule récompense (négative). Si cela est possible pour du maintien de voie sans autres objects dynamiques, cela est impossible à réaliser dans des cas plus complexes incluant d'autres véhicules ou des piétons pour des raisons de sécurité. Finalement, leur réseau de neurones a été optimisé pour être le plus petit possible mais quand même capable d'apprendre leur tâche très simple rapidement. Mais comme leur réseau possède très peu de paramètres, il n'aura pas la capacité d'apprendre des cas réels plus complexes. Et si l'on rajoute des paramètres au réseau, le temps d'apprentissage en deviendra beaucoup plus long et rendra l'apprentissage directement dans le monde réel quasiment impossible.

C'est pour toutes ces raisons que nous pensons que bien que cet article était extrêmement novateur, il semble quasiment impossible à étendre à des cas d'applications plus complexe et n'est pas utilisable pour notre cas : la conduite en environnement urbain.

7.2.2 VISTA : Renforcement avec simulateur fondé sur des images réelles

L'autre article appliquant de l'apprentissage par renforcement sur une voiture réelle est le travail de Amini et al. [2020]. Ce travail cherche lui aussi uniquement à contrôler l'angle au volant pour le maintien de voie. Cependant, il possède des différences fondamentales avec le travail présenté précédemment. Tout d'abord, Amini et al. [2020] ont réussi à transférer l'apprentissage de la simulation au monde réel sans aucun réapprentissage sur le véhicule réel. La contribution principale de Amini et al. [2020] a été

d'utiliser un simulateur fondé sur des images réelles [Bojarski et al., 2016, Toromanoff et al., 2018] pour effectuer de l'apprentissage par renforcement et non pas seulement pour évaluer en boucle fermée les agents appris par imitation. Ils ont aussi eu l'idée d'utiliser une estimation de la profondeur pour effectuer de meilleures augmentations de point de vue. Cette idée permet en effet d'obtenir la position 3D des pixels de l'image sans avoir besoin de faire d'hypothèse additionnelle comme l'hypothèse sol/ciel (cf Chapitre 3 section 6.3.1) par exemple. Cette estimation de profondeur se sert des travaux de l'état de l'art sur l'estimation de profondeur à partir d'une caméra monoculaire [Godard et al., 2017]. L'architecture de leur simulateur fondé sur des images réelles peut se trouver en Figure 7.1.

De plus, les chercheurs ont appris sur des routes bien plus diversifiées, ce qui permet une meilleure capacité de généralisation que le travail présenté précédemment [Kendall et al., 2019]. En effet, tout l'apprentissage se faisant en simulation, il est beaucoup plus aisé d'utiliser des bases de données bien plus importantes et bien plus variées que lorsque l'apprentissage se fait directement sur le véhicule réel avec les reprises en main du conducteur humain comme seule récompense [Kendall et al., 2019].

Finalement, Amini et al. [2020] ont aussi effectué une comparaison de leur approche par renforcement avec différentes méthodes de l'état de l'art, en particulier une comparaison avec l'apprentissage par imitation avec des augmentations de labels comme notre propre travail [Toromanoff et al., 2018] ou le travail de Nvidia [Bojarski et al., 2016]. Ils ont aussi effectué une comparaison avec des méthodes de transfert d'apprentissage d'un simulateur au monde réel [Tobin et al., 2017, Bewley et al., 2019] en utilisant CARLA comme simulateur. Ces expériences sont présentées en Figure 7.2. Leurs résultats montrent que les méthodes de transfert d'apprentissage de CARLA au monde réel ne sont pas encore assez matures et aboutissent à de nombreuses reprises en main lorsqu'elles sont évaluées sur le véhicule réel. Cela rejoint nos arguments émis dans le Chapitre 2 (Section 2.5.1.2), les simulateurs comme CARLA [Dosovitskiy et al., 2017] ou Airsim [Shah et al., 2018] ne sont probablement pas encore assez aboutis pour réellement réussir à transférer un apprentissage dans la simulation au monde réel. L'approche par imitation permet d'atteindre des performances raisonnables avec peu de reprises en main. Cependant, c'est leur propre approche par renforcement en utilisant leur simulateur VISTA qui atteint les meilleures performances possibles, i.e. aucune reprise en main n'a été nécessaire lors de l'évaluation sur le véhicule réel.

Ce travail est extrêmement prometteur, Amini et al. [2020] ont en fait prouvé que l'apprentissage par renforcement pouvait atteindre des meilleures performances que l'apprentissage par imitation sur un véhicule réel. Il est important de noter qu'ils ont utilisé un algorithme de renforcement fondé sur la politique relativement basique [Williams, 1992] et que leur fonction de récompense est la plus simple qui soit, une récompense de 1 à chaque pas de temps sauf quand l'agent s'éloigne trop de sa voie où il reçoit une récompense de 0 et l'épisode se termine. Cela est encore une preuve supplémentaire de l'intérêt de l'apprentissage par renforcement, même un algorithme simple et une fonction de récompense triviale peuvent atteindre des performances meilleures que l'apprentissage par imitation. Nous verrons dans la suite comment nous avons utilisé les idées introduites

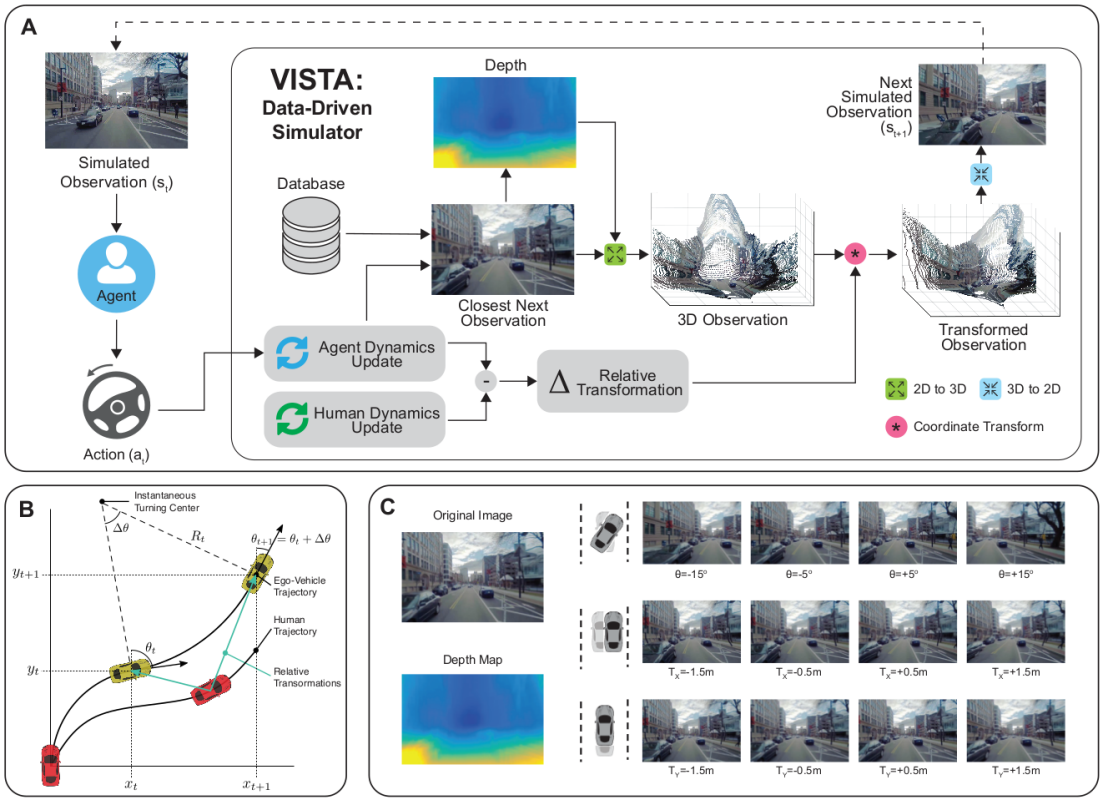


FIGURE 7.1 – Schéma du simulateur VISTA (pour *Virtual Image Synthesis and Transformation for Autonomy*) fondé sur des images réelles introduit par Amini et al. [2020]. Leur simulateur est en fait très similaire au nôtre introduit dans le Chapitre 6 et on retrouve bien les deux modèles physiques de l'agent et du conducteur humain permettant de déterminer une position relative qui est elle-même utilisée pour simuler le point de vue qu'aurait l'agent si il avait conduit. La différence fondamentale se trouve au niveau de la génération de point de vue, en effet Amini et al. [2020] utilise l'estimation de profondeur d'un réseau de neurones pour déterminer la position 3D des pixels de l'image. Cela permet d'éviter de faire l'hypothèse sol/ciel générant des artefacts sur les objets ne vérifiant pas cette hypothèse. Cependant, cette estimation de profondeur amène elle aussi des artefacts car elle sera nécessairement bruitée. L'autre différence fondamentale est que les auteurs utilisent ce simulateur pour entraîner un agent par renforcement et non pas seulement pour évaluer en boucle fermée les agents appris par imitation. Image provenant de Amini et al. [2020].

dans ce travail pour effectuer de l'apprentissage par renforcement sur données réelles pour une application bien plus complexe de conduite en environnement urbain.

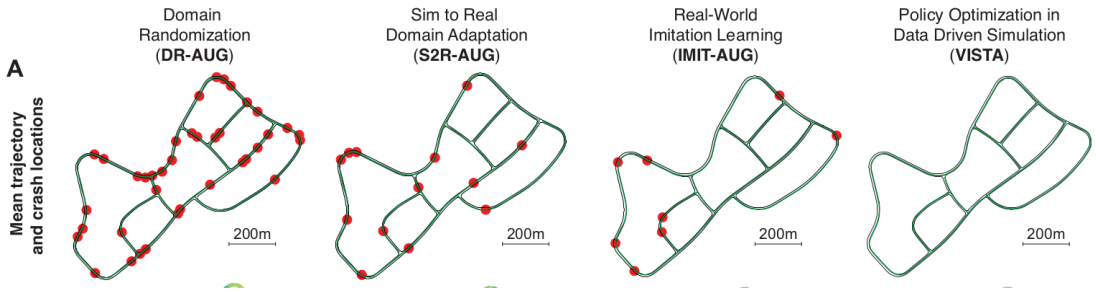


FIGURE 7.2 – Résultats sur véhicule réel de quatre approches différentes pour le contrôle latéral pour du maintien de voie. Chaque approche est évaluée sur exactement les mêmes routes et les points rouges correspondent aux reprises en main du conducteur humain. De gauche à droite, le transfert par randomisation de texture [Tobin et al., 2017], l'adaptation de domaine simulation au réel [Bewley et al., 2019], l'apprentissage par imitation avec génération de point de vue et label associé [Bojarski et al., 2016, Toromanoff et al., 2018] et finalement l'approche par renforcement de Amini et al. [2020]. Les deux premières méthodes de transfert de simulation (CARLA) au monde réel ont des performances bien moindres que l'apprentissage par imitation sur données réelles avec génération de point de vue. Finalement, les meilleurs performances (aucune reprise en main nécessaire) sont atteintes avec leur apprentissage par renforcement sur leur simulateur fondé sur des images réelles. Image provenant de Amini et al. [2020].

7.3 Génération de point de vue avec estimation de profondeur

L'idée d'estimer la profondeur pour générer de meilleures augmentations de point de vue semble en fait peu utile pour l'application de maintien de voie utilisée dans l'article VISTA [Amini et al., 2020]. En effet, il n'y a aucun objet 3D pertinent dans la scène et l'hypothèse sol/ciel semble suffisante pour ce cas d'application, ce qui sera vérifié dans la suite par nos propres travaux pour le contrôle latéral (cf Section 7.4). En fait, nous pourrions appliquer de l'apprentissage par renforcement sur notre propre simulateur sans estimation de profondeur et cela conduirait très probablement aux mêmes résultats que Amini et al. [2020]. Par contre, cette profondeur peut s'avérer extrêmement utile si on considère une application où il y a réellement des objets 3D pertinents dans la scène. En particulier, si l'on veut contrôler le véhicule pour freiner ou éviter des objets dynamiques comme des voitures ou des piétons, l'hypothèse sol/ciel s'avère complètement fausse. Notre idée initiale était donc de reprendre le principe du simulateur introduit par VISTA mais de l'appliquer pour le contrôle longitudinal et latéral avec des objets dynamiques.

Dans cette section, nous allons donc présenter notre simulateur fondé sur des données réelles qui sera utilisé pour apprendre par renforcement le contrôle latéral et longitudinal en environnement urbain. D'abord nous décrirons notre estimation de profondeur utilisant une seule caméra fisheye. Ce travail a en fait été réalisé par d'autres personnes à Valeo et publié dans un article à la conférence ICRA 2020 [Kumar et al., 2020]. Dans

un second temps, nous présenterons comment nous avons utilisé cette estimation de profondeur pour générer des images décalées de la position du conducteur humain. Nous verrons que notre reprojection est différente de la majorité des travaux utilisant une profondeur pour faire de la génération de point de vue car nous avons la profondeur sur l'image initiale tandis que ces travaux ont la profondeur sur l'image cible.

7.3.1 Estimation de profondeur à partir d'une seule caméra fisheye : FisheyeDistanceNet

L'article FisheyeDistanceNet [Kumar et al., 2020] s'inspire du travail de Zhou et al. [2017] qui cherche à estimer la profondeur à partir d'une seule caméra pinhole standard. L'intérêt majeur de ces deux travaux est qu'ils ne nécessitent aucune annotation, ils n'ont en fait besoin que d'images en séquences. L'idée introduite par Zhou et al. [2017] est d'abord d'estimer un déplacement, i.e. une translation et une rotation, entre deux images consécutives. Ensuite, leur réseau de neurones va aussi estimer une carte de profondeur de l'image courante. Finalement, ils utilisent un algorithme de projection différentiable qui estime l'image courante en prenant en entrée l'image précédente, la carte de profondeur estimée de l'image courante ainsi que le déplacement entre l'image précédente et l'image courante. Cet algorithme de projection est illustré sur la Figure 7.3. Pour chaque pixel de l'image courante (l'image *cible*), on projette ce pixel dans le repère de l'image précédente (l'image *source*) en utilisant la profondeur et le déplacement estimés par le réseau. Ensuite, on peut estimer l'image cible en effectuant une interpolation sur les pixels de l'image source.

Finalement, Zhou et al. [2017] ont construit une fonction de coût dépendant de la distance entre l'image courante estimée par cet algorithme et la véritable image courante. Comme tout le processus est différentiable, ils peuvent bien rétropropager la fonction de coût dans tous les composants du réseau de neurones, i.e. l'estimation du déplacement ainsi que l'estimation de la carte de profondeur. Toute cette architecture est représentée sur la Figure 7.4.

Cette approche permet ainsi d'entraîner un réseau pour estimer le déplacement et surtout pour estimer la profondeur seulement à partir d'une séquence d'images sans aucune autre annotation que l'ordre des images dans la séquence. Cependant, cette approche s'appuie en fait implicitement sur plusieurs hypothèses. La première et la plus impactante est que la scène doit être totalement statique sans aucun objet mobile autre que la caméra qui enregistre les différentes images de la séquence. En effet, la reprojection estimant l'image courante admet implicitement que les pixels de l'image ne bougent pas entre l'image source et l'image cible. Cela signifie que les données utilisées pour l'entraînement doivent contenir le moins possible d'objets en mouvement. L'autre hypothèse effectuée par cette approche est qu'il n'y a aucune occultation entre la vue de l'image source et la vue de l'image cible ce qui est nécessairement faux dès que des objets 3D sont présents dans la scène et qu'on cherche à estimer un pixel derrière cet objet. Lorsque ces hypothèses sont enfreintes, le gradient provenant de la fonction de coût peut ralentir voire inhiber l'apprentissage. Pour améliorer la robustesse de leur méthode, Zhou et al. [2017] entraînent un *réseau d'explicabilité* (simultanément avec les

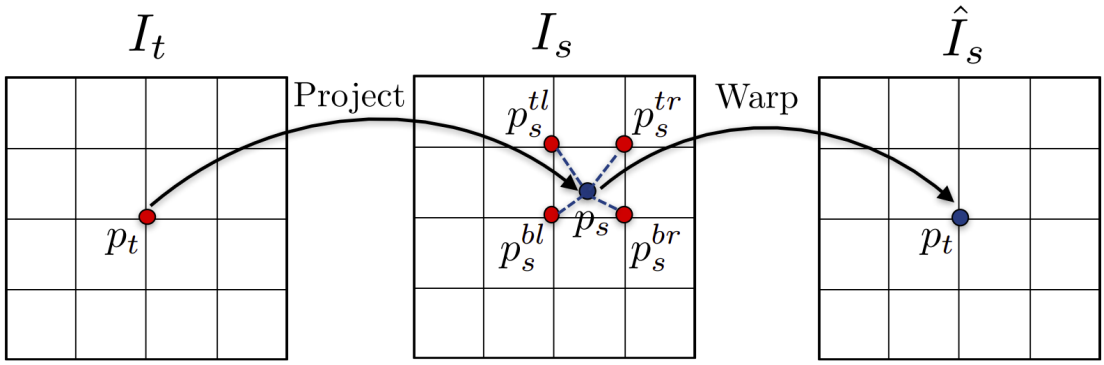


FIGURE 7.3 – Illustration de l'algorithme différentiable de projection d'image introduit par Zhou et al. [2017]. Pour chaque pixel p_t de l'image cible I_t (l'image courante), ce pixel est d'abord projeté dans le repère de l'image source I_s (l'image précédente) en utilisant la profondeur et le déplacement estimés par le réseau. Ensuite, une interpolation bilinéaire est utilisée pour obtenir la couleur du pixel à la position p_t dans l'image source déformée $\hat{I}_s$. L'idée est que cette image source déformée devrait être égale à la véritable image cible à condition que la profondeur et le déplacement estimés par le réseau soient corrects. Ainsi, en utilisant une fonction de coût dépendant de la distance entre l'image cible estimée et la véritable image cible, le réseau apprend à estimer la pose et la profondeur de l'image courante. Image provenant de Zhou et al. [2017].

réseaux qui estiment la pose et la profondeur) qui calcule un poids pour chaque pixel de l'image cible. L'idée est que le réseau peut apprendre de lui-même à masquer les pixels qui sont impossibles à estimer correctement soit parce qu'il y a un objet mobile dans la scène, soit parce qu'il y a une occultation entre les deux vues cible et source. Des exemples qualitatifs de *masque d'explicabilité* sont illustrés sur la Figure 7.5. Ces résultats sont très intéressants car on voit que le réseau estime globalement correctement quelles parties de l'image ne doivent pas être prises en compte dans la fonction de coût de reconstruction, que ce soit parce que ces pixels appartiennent à des objets mobiles ou à cause d'occultations par la végétation par exemple.

Cette troisième partie du réseau de neurones qui pondère chaque pixel de l'image dans la fonction de coût permet ainsi de réduire l'impact négatif que pourrait avoir les images d'entraînement pour lesquels les deux hypothèses ne sont pas vérifiées. Cependant, comme nous le verrons par la suite, cela ne permet pas non plus de complètement résoudre le problème, et si la base d'entraînement comporte trop d'objets mobiles par exemple, les performances en seront grandement détériorées.

Tous les principes de l'article de Zhou et al. [2017] décrits précédemment ont été repris par Kumar et al. [2020] pour la même application mais sur des séquences d'images fisheye provenant des capteurs Valeo. Des améliorations ont été apportées pour augmenter encore les performances ainsi que pour mieux prendre en compte les déformations induites par les images fisheye. Par exemple, Kumar et al. [2020] utilisent une combinaison de réseau de super-résolution [Shi et al., 2016] et de couches de convolution déformables

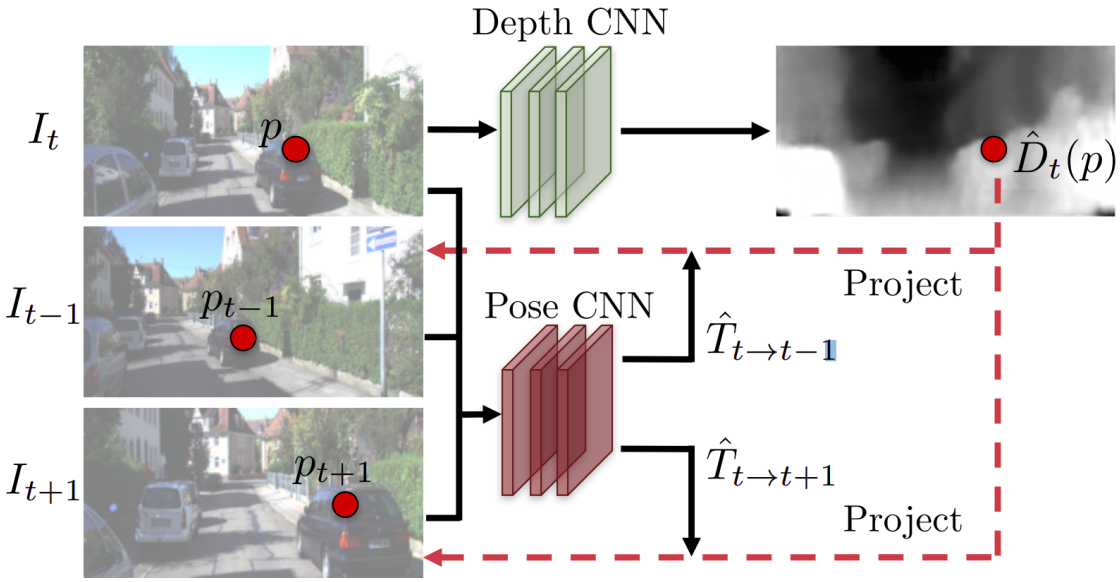


FIGURE 7.4 – Structure de l'apprentissage non supervisé d'une carte de profondeur à partir d'une séquence d'images introduite par Zhou et al. [2017]. L'idée est d'abord d'estimer une carte de profondeur pour l'image courante I_t ainsi que le déplacement entre l'image précédente I_{t-1} (ou l'image suivante I_{t+1}) et l'image courante. Puis en utilisant la carte de profondeur estimée, l'image précédente (ou l'image suivante) et le déplacement estimé entre ces deux images, ils peuvent utiliser une reprojection pour estimer l'image courante. Une fonction de coût entre la véritable image courante et l'image courante estimée grâce à cette reprojection peut ainsi être rétro-propagée dans tout le réseau de neurones, i.e. la partie qui estime le déplacement ainsi que celle qui estime la profondeur. En effet, ce coût sera nul à condition que le déplacement et la carte de distance estimés soient corrects. Image provenant de Zhou et al. [2017].

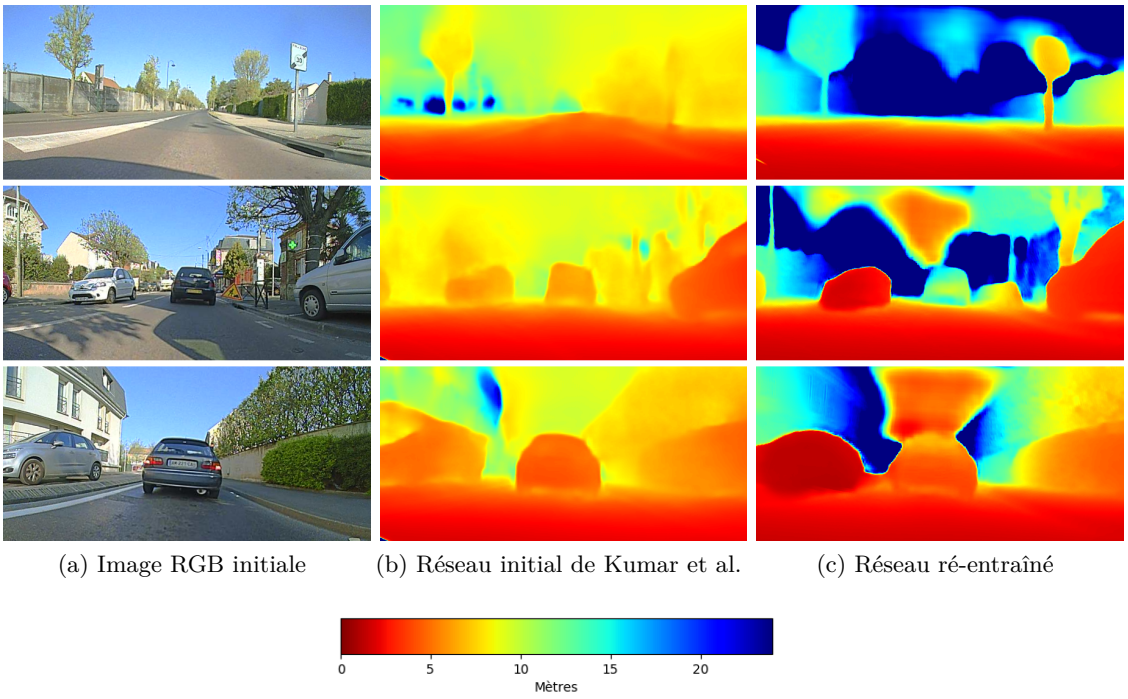
[Zhu et al., 2019] qui permettent de mieux gérer les déformations des images fisheye. Finalement, Kumar et al. [2020] ont aussi rajouté une fonction de coût à leur entraînement qui incite le réseau à estimer des profondeurs cohérentes sur plusieurs pas de temps plutôt que juste sur deux pas de temps successifs.

De plus, comme indiqué précédemment cette approche a été proposée et développée par des ingénieurs à Valeo ce qui a grandement facilité le transfert de code et de connaissances. Par exemple, nous avons pu directement utiliser leur réseau entraîné pour estimer des cartes de profondeur sur notre propre base de données. Nous avons aussi essayé de ré-entraîner un modèle spécifiquement sur nos données mais les performances sur les objets mobiles comme les autres véhicules étaient mauvaises. En particulier, les véhicules devant étaient estimés trop loin et les véhicules en sens inverse sur l'autre voie étaient au contraire estimés beaucoup trop proches (cf Figure 7.6). Cela est dû au fait que l'hypothèse de scène statique est totalement fausse pour la grande majorité des véhicules de notre base de données. En effet, les véhicules devant vont généralement à la

FIGURE 7.5 – Exemple de masques d'explicabilité calculés par le réseau de Zhou et al. [2017]. L'idée est d'estimer les pixels de l'image source qui ne respectent pas les hypothèses nécessaires à la fonction de coût de reprojection, i.e. une scène statique sans objets dynamiques ainsi que l'absence d'occultation entre les deux vues source et cible. On constate que le réseau arrive bien à estimer quelles parties de l'image représentent des objets dynamiques (3 premières lignes) ou bien les parties de l'image source susceptibles d'apporter des occultations dans l'image cible comme la présence de végétation (2 dernières lignes). Image provenant de Zhou et al. [2017].

même vitesse que le conducteur, ils ne se rapprochent donc pas d'une image à l'autre. En fait, pour l'algorithme, c'est comme si l'objet était à l'infini et il estime donc sa distance à l'infini. C'est exactement l'effet inverse pour les véhicules qui se dirigent vers nous, l'objet se rapproche beaucoup plus que prévu : l'algorithme estime donc ces véhicules bien plus proches qu'ils ne le sont en réalité. Nous avons cependant noté une amélioration des performances sur toutes les parties de l'image réellement statiques comme la route, le trottoir, les lignes et nous avons donc utilisé notre propre estimation de profondeur pour le contrôle latéral. Pour le contrôle longitudinal où les autres véhicules sont une

information très importante, nous avons utilisé le réseau initial de Kumar et al. [2020]. En effet, leur base de données a été choisie spécifiquement pour comporter le moins d'objets mobiles possible et leur réseau estime donc bien mieux la profondeur des autres véhicules.



(a) Image RGB initiale (b) Réseau initial de Kumar et al. (c) Réseau ré-entraîné

FIGURE 7.6 – Exemple d'estimation de profondeur du réseau initial de Kumar et al. [2020] et de notre réseau ré-entraîné sur notre propre base de données. On voit bien sur la première ligne que notre réseau estime mieux la profondeur sur les objets statiques. Il est donc plus approprié pour le contrôle latéral, où les objet statiques comme le trottoir sont les plus pertinents à la conduite. Cependant, sur la deuxième et troisième lignes, on constate que notre réseau estime le véhicule devant nous beaucoup trop loin (quasiment à l'infini) et estime le véhicule venant vers nous beaucoup trop proche. Le réseau initial de Kumar et al. [2020] a des estimations globalement plus bruitées, notamment sur les objets statiques, mais a de bien meilleures performances sur les objets mobiles et semble donc plus approprié pour le contrôle longitudinal.

Des exemples qualitatifs d'estimation de carte de profondeur sont illustrés sur la Figure 7.6. En particulier, on voit bien les cas d'échecs de notre réseau, i.e. la voiture devant nous est estimée presque à l'infini alors que les voitures dans le sens inverse sont estimées beaucoup trop proches. Le réseau initial de Kumar et al. [2020], bien qu'ayant des estimations globalement plus bruitées, a de bien meilleures performances sur ces objets mobiles et semble donc plus approprié pour le contrôle longitudinal.

7.3.2 Projection à partir d'une carte de profondeur et de l'image RGB

Maintenant que nous avons une carte de profondeur (bruitée) automatique pour chacune des images de notre base de données, nous voulons générer des points de vue plus réalistes que ceux utilisés par notre simulateur dans le chapitre précédent. L'idée principale est de ne plus faire l'hypothèse sol/ciel et d'utiliser une estimation de la profondeur pour mieux gérer les objets en 3D de la scène. Nous avons donc en entrée, l'image source, la carte de profondeur de l'image source et la transformation que l'on veut simuler. Cela semble identique à la projection précédente, pourtant un détail rend notre reprojection totalement différente de celle présentée précédemment. En fait, nous avons maintenant la carte de distance sur l'image source et non pas sur l'image cible. Nous allons voir que ce détail est en fait une différence fondamentale. Implémenter cette reprojection nous a en effet demandé quasiment un mois de développement entier. Nous allons donc maintenant présenter pourquoi un tel changement et quelles sont les difficultés que nous avons dû résoudre. Avant tout chose, il est important de comprendre la différence fondamentale entre la *transformation directe* (en anglais *forward warping*) : on projette chaque pixel de l'image source vers l'image cible et la *transformation inverse* (en anglais *inverse warping*) : pour chaque pixel de l'image cible on cherche quels pixels de l'image source lui correspondent. Ces deux méthodes sont illustrées sur la Figure 7.7.

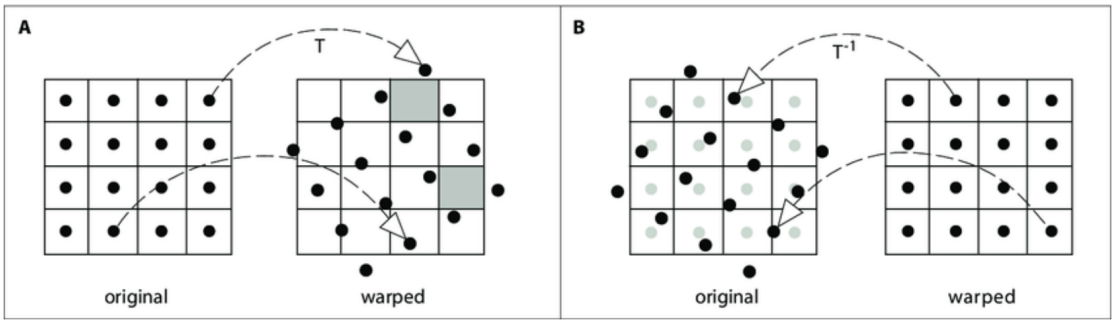


FIGURE 7.7 – À gauche, une transformation directe. Certains pixels de l'image source restent vide car aucun pixel de l'image source ne se projetent à ces positions. À droite, une transformation inverse. On calcule les couleurs à attribuer aux pixels de l'image cible en effectuant une interpolation sur les pixels de l'image source. De manière générale, on utilise une transformation inverse si cela est possible. Image provenant de Schwarz [2007].

Dans le cas précédent pour l'estimation de profondeur, on a un nuage de points 3D dans l'espace cible et l'on veut colorer ce nuage de points à partir des pixels d'une image source. Ce nuage de points est parfaitement synchronisé avec l'image cible, i.e il y a exactement autant de points 3D que de pixels dans l'image et chaque point 3D correspond à un unique pixel de l'image. Cela signifie qu'on fait une transformation inverse.

Dans notre cas actuel, notre nuage de points 3D est déjà coloré et est synchronisé avec l'image source, mais pas avec l'image cible. Ainsi, certains pixels de l'image cible ne seront associés à aucun point 3D : nous effectuons en effet une transformation directe.

Pour les pixels de l'image cible n'ayant été associés à aucun point 3D, nous comblons ces pixels vides en fonction de la couleur des pixels voisins (*image inpaiting* en anglais).

Cependant, un problème majeur reste à résoudre, le problème de *grossissement des points 3D* qui survient lorsque le point de vue simulé se rapproche d'un objet de la scène. En effet, supposons que dans l'image source, il y ait un véhicule situé 3 mètres devant. Disons que ce véhicule représente 20% des pixels de l'image source : il y aura donc 20% des points 3D qui porterons la couleur de ce véhicule. Imaginons maintenant que l'on veut simuler un rapprochement de 2 mètres, le véhicule sera tellement proche de nous qu'il devrait représenter quasiment 100% des pixels de l'image cible (cf Figure7.8). Pourtant, il n'y aura que 20% des points 3D qui porteront la couleur de ce véhicule, il n'y aura donc dans l'image cible que 20% des pixels qui représenteront la voiture si on utilise seulement une reprojection naïve associant chaque point 3D à un seul pixel.

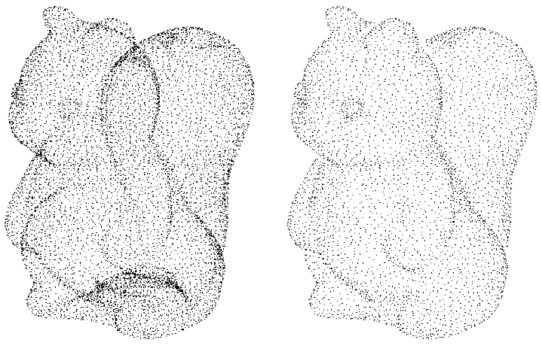
En fait, lorsque l'on simule un rapprochement d'un objet avec une reprojection naïve, les points 3D s'écartent de plus en plus au fur à mesure que l'on se rapproche de l'objet et cela donne l'illusion d'un *objet fantôme* comme illustré en Figure 7.8. De plus, ce problème ne peut pas être résolu simplement en comblant les pixels vides car ces pixels seront généralement associés à des pixels de l'arrière plan (comme le ciel par exemple) et ne resteront pas sans couleur.

(a) Image initiale et profondeur estimée (b) Zoom simulé de 1m (c) Zoom simulé de 2m (d) Zoom simulé de 3m

FIGURE 7.8 – Exemple d'une reprojection naïve (première ligne) comparée à une reprojection qui grossit les pixels en fonction de leur distance dans la vue simulée (2ème ligne). Dans le premier cas, on a l'illusion d'un *objet fantôme*, on distingue la silhouette générale du véhicule devant mais la grande majorité des pixels sont en fait des pixels de l'arrière plan comme le ciel. Plus on simule un point de vue rapproché, moins l'objet est visible : dans notre exemple, avec un rapprochement de 3 mètres, on n'arrive presque plus à distinguer le véhicule devant. Dans le deuxième cas, l'objet est bien visible car on a fait en sorte de grossir les pixels lui correspondant et on ne voit pas l'arrière plan à travers. Il est important de noter que notre profondeur comporte nécessairement des petites erreurs (estimation par un réseau de neurones) et que ces erreurs, mêmes si elles sont faibles, peuvent générer des artefacts conséquents sur le résultat final, particulièrement lorsqu'on simule un zoom très rapproché d'un objet (cf Zoom simulé de 3m).

En fait, notre problème appartient au domaine de l'infographie et plus spécifiquement

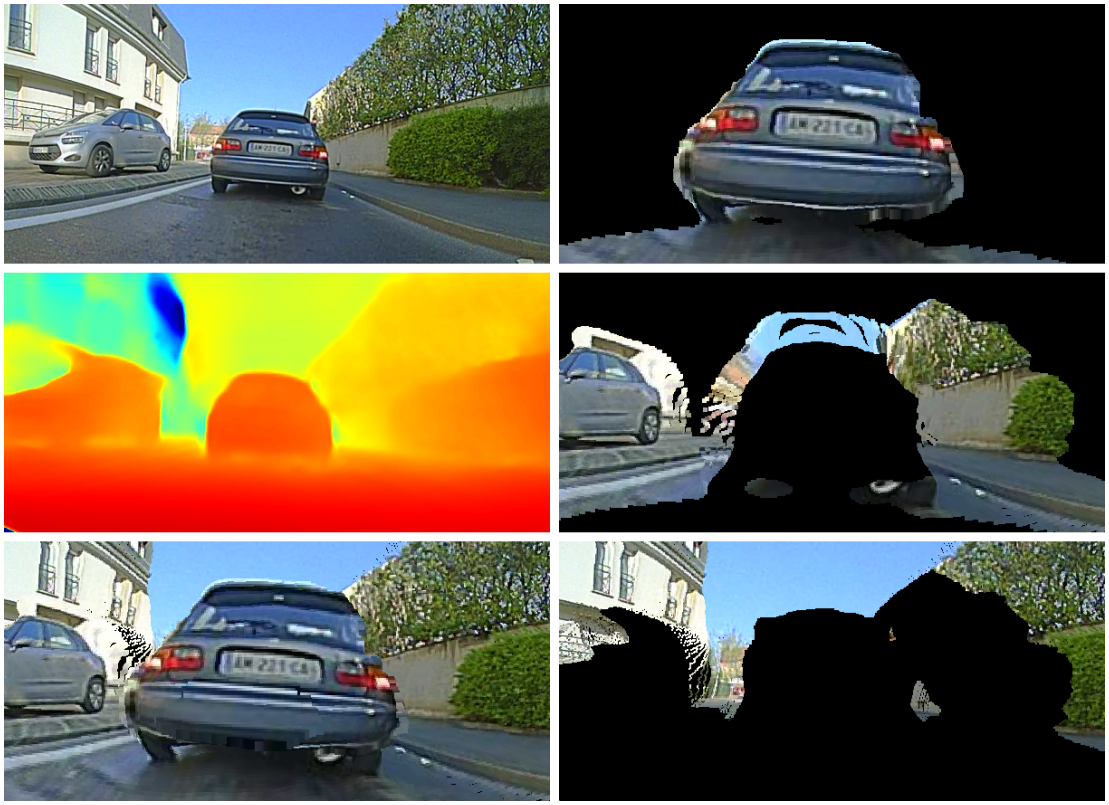
à l'application de *l'affichage de points 3D sur des images* (en anglais *image-based rendering of point cloud*) [Chang and Guo-Ping, 2019] et du problème du *masque des points 3D cachés* (en anglais *hidden point removal*) [Katz et al., 2007, Katz and Tal, 2015]. Un exemple d'une telle application est illustrée sur la Figure 7.9. En effet, notre problème peut se présenter sous la forme d'un nuage de points 3D colorés que nous voulons afficher depuis différents points de vue. Ainsi, nous devons calculer quels points du nuage ne sont plus visibles dans ces différents points de vue et aussi en quelque sorte grossir les points 3D les plus proches, i.e. projeter ces points dans plusieurs pixels de l'image cible. Il est important de noter que contrairement aux approches de l'état de l'art, notre profondeur n'est qu'une estimation provenant d'un réseau de neurones et comporte des erreurs qui peuvent générer des artefacts conséquents dans le résultat final, en particulier lorsqu'on simule un zoom très rapproché (cf Zoom simulé de 3m sur la Figure 7.8).



(a) Input (b) Output

FIGURE 7.9 – À gauche, le nuage de points 3D initial à visualiser. À droite, le résultat de l'algorithme de Katz and Tal [2015], l'idée principale est de calculer quels points sont visibles à partir d'un certain point de vue. Image provenant de Katz and Tal [2015].

Par manque de temps et d'expérience dans ce domaine particulier, nous avons implémenté notre propre méthode simple pour générer nos différents points de vue en s'inspirant des différents travaux de l'état de l'art du domaine en particulier l'article de Bouchiba et al. [2018]. Notre méthode construit notre point de vue final à partir d'une pyramide d'images intermédiaires comme illustré en Figure 7.10. Notre idée est de d'abord projeter les points 3D les plus rapprochés du point de vue que l'on veut simuler. Comme ces points 3D sont très proches, nous les projetons avec un voisinage large dans l'image cible. L'intuition étant que plus un point 3D est proche, plus il représentera de pixel dans l'image final. Nous construisons d'autres images intermédiaires en projetant à chaque itération les points 3D restants (i.e. les points qui n'ont pas encore été projetés) les plus proches. Finalement, nous construisons notre image finale en accumulant toutes les images intermédiaires ainsi obtenues. Notre approche est illustrée sur la Figure 7.10. En fait, notre approche s'inspire de l'approche de Bouchiba et al. [2018] qui utilise aussi en quelque sorte une reconstruction pyramidale de l'image final en projetant d'abord les points 3D du plus proche au plus éloigné.



(a) Image initiale et résultat final (b) Images intermédiaires

FIGURE 7.10 – Illustration de notre approche pour visualiser notre nuage de point 3D initial avec un autre point de vue : ici un zoom de 2 mètres. Tout d'abord, nous projetons les points 3D les plus proches dans le point de vue simulé (image en haut à droite). Comme ces points 3D sont très rapprochés dans cette nouvelle vue, nous les projetons sur plusieurs pixels en utilisant un voisinage très large. Nous projetons ensuite les points 3D restants les plus proches sur une deuxième image intermédiaire (image au milieu à gauche). Ces points étant moins proche, nous les projetons avec un voisinage plus faible. Finalement, nous projetons tous les points restants (image en bas à droite). Il est important de noter que de nombreux artefacts sont présents car notre estimation de profondeur est très bruitée. En particulier, le réseau a tendance à lisser la carte de profondeur et évite d'estimer de trop grande discontinuité : l'arrière plan autour des objets 3D est généralement estimé trop proches. Finalement, nous obtenons le résultat final (image en bas à gauche) en accumulant les images intermédiaires. Notre approche s'inspire de la méthode de Bouchiba et al. [2018].

Dans cette section, nous avons décrit comment nous générons nos augmentations de point de vue à partir d'une estimation d'une carte de profondeur de nos images. Cela permet donc d'obtenir un simulateur comme celui utilisé dans l'article de Amini et al.

[2020] dans lequel nous pouvons maintenant entraîner des agents par renforcement. Dans la section suivante, nous allons d'abord décrire notre approche pour le contrôle latéral qui peut se voir comme une extension des travaux de Amini et al. [2020] à un cas d'application plus complexe. Dans la section d'après, nous présenterons nos travaux en cours pour l'apprentissage par renforcement du contrôle longitudinal.

7.4 Renforcement pour le contrôle latéral en environnement urbain

Dans cette section, nous décrirons notre méthode pour effectuer de l'apprentissage par renforcement pour le contrôle latéral en environnement urbain. Notre approche est en fait une combinaison de plusieurs travaux précédents, tout d'abord notre propre travail sur CARLA [Toromanoff et al., 2020] introduisant les *indices implicites* décrit dans le Chapitre 5, l'approche de VISTA [Amini et al., 2020] en particulier pour l'idée de génération des points de vue à partir d'une estimation de la profondeur et finalement notre propre article d'apprentissage par imitation pour le contrôle latéral [Toromanoff et al., 2018] décrite dans le chapitre précédent.

7.4.1 Apprentissage d'indices implicites pour le contrôle latéral

Tout d'abord nous avons ré-utilisé l'idée des *indices implicites* qui permet d'entraîner des réseaux par renforcement avec bien plus de paramètres et des entrées de plus grande dimension. Nous avons utilisé seulement deux tâches comme indices : la segmentation sémantique ainsi que l'imitation. L'estimation d'une segmentation sémantique poursuit le même but que pour notre travail sur CARLA et l'imitation est en fait une tâche très proche de notre objectif final et semble donc particulièrement pertinente à ajouter comme indice. Pour l'estimation de la segmentation sémantique, nous avons utilisé une base de données réelles interne à Valeo comportant environ 40 000 images avec leurs segmentations sémantiques annotées associées.

Pour ce premier apprentissage, nous avons utilisé les mêmes augmentations de labels que celles utilisées dans le chapitre précédent. En fait, cet entraînement comporte seulement quelques différences avec notre travail publié à la conférence IROS 2018. Premièrement, l'augmentation de point de vue n'utilise pas une hypothèse sol/ciel mais utilise une estimation de profondeur comme indiqué dans la section précédente. Cela est en fait exactement l'approche prise par l'article VISTA [Amini et al., 2020]. Deuxièmement, nous utilisons l'estimation d'une carte de segmentation sémantique comme tâche auxiliaire dans l'idée que cette tâche permette d'accélérer l'entraînement et d'apporter un signal d'apprentissage pertinent. Finalement, nous utilisons un réseau avec plusieurs têtes pour gérer différentes commandes de navigation haut niveau : tout droit, à droite ou à gauche. Cela permet de ne plus retirer les images où le conducteur humain change de voie ou tourne dans une intersection mais plutôt de leur associer une commande de navigation différente. Ainsi, notre agent est capable de changer de voie ou bien d'apprendre à agir à une intersection pour se diriger vers la bonne voie en fonction de la commande de

navigation donnée. Cela permet de gérer un cas d'application bien plus large que notre approche précédente ou que l'application de VISTA. L'architecture générale de notre apprentissage supervisé pour le contrôle latéral est illustrée en Figure 7.11.

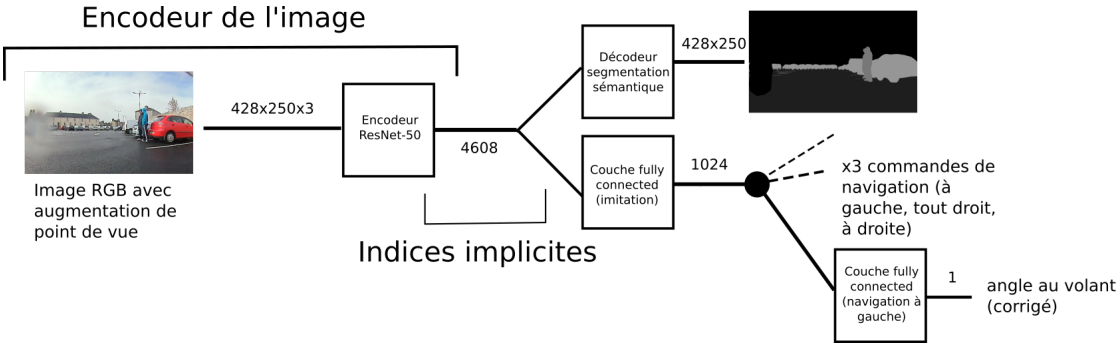


FIGURE 7.11 – Entraînement de nos indices implicites pour le contrôle latéral avec les deux fonctions de coût que l'on a utilisées. Nous avons choisi d'utiliser deux indices pour ce premier apprentissage, l'estimation d'une carte de segmentation ainsi que l'imitation de l'angle au volant pris par le conducteur humain. Il est important de noter que nous avons généré des augmentations de point de vue avec leur label associé de la même manière que pour notre travail présenté dans le Chapitre 6. Nous avons aussi rajouté des commandes de navigation pour permettre à l'imitation de s'adapter en fonction de la situation lorsque le conducteur humain change de voie ou se trouve sur une intersection. Comme pour notre travail sur CARLA [Toromanoff et al., 2020], l'idée est ensuite de fixer les poids de l'encodeur et d'utiliser les features intermédiaires comme état pour l'agent appris par renforcement.

De plus, un intérêt majeur d'entraîner l'imitation durant cette première phase est que cela permet d'avoir une référence à comparer à notre agent final appris par renforcement, surtout que l'on sait que cette référence est capable de généralisation et atteint de bonne performance sur le véhicule réel. Cette comparaison a d'ailleurs aussi été faite dans l'article VISTA [Amini et al., 2020] afin de mesurer l'intérêt de l'apprentissage par renforcement par rapport à l'apprentissage par imitation.

7.4.2 Apprentissage par renforcement sur simulateur fondé sur des images réelles

L'idée est maintenant de réellement entraîner notre agent par renforcement dans le simulateur. Tout comme pour notre travail sur CARLA, nous utilisons les indices implicites comme état. Nous utilisons aussi notre propre algorithme de renforcement fondé sur la fonction de valeur, Rainbow IQN Ape-X [Toromanoff et al., 2019], présenté dans le Chapitre 4. Notre approche diffère donc de l'approche de VISTA car nous utilisons une image d'entrée et un réseau beaucoup plus grand grâce à l'utilisation d'un pré-entraînement des indices implicites. Notre réseau est donc potentiellement plus expressif et capable d'apprendre des tâches plus complexes. De plus, notre algorithme de renfor-

cement est bien plus élaboré ce qui devrait permettre d'apprendre plus rapidement et d'obtenir de meilleures performances finales. Cependant, nous ne pouvons pas réellement nous comparer à Amini et al. [2020] car nous n'avons pas accès ni aux mêmes données d'entraînement, ni aux mêmes capteurs, ni aux mêmes véhicules.

Dans cette section, nous allons présenter la configuration de notre apprentissage par renforcement, en particulier quel est l'espace d'états et d'actions que l'on a choisi et comment nous avons construit notre fonction de récompense.

Espace d'états Pour ajouter un aspect temporel à l'entrée de notre agent, nous avons choisi de concaténer trois indices implicites consécutifs comme état de l'environnement.

Espace d'actions Nous avons utilisé 57 actions différentes, chaque action étant la dérivée de l'angle au volant que l'on veut appliquer. Nous avons choisi de faire calculer par l'agent la dérivée de l'angle au volant pour être plus réaliste et éviter des changements trop importants et trop rapides du volant dans le véhicule réel. En effet, ces changements brusques étaient possibles dans CARLA mais ne le sont pas sur un véhicule réel, sans parler de question de confort.

Fonction de récompense Notre récompense est globalement très similaire à notre récompense de position désirée dans CARLA (cf Chapitre 5 Section 5.4.2). Plus l'agent s'éloigne de sa voie, plus la récompense diminue. Nous avons aussi trouvé que l'ajout d'une récompense sur l'angle était particulièrement importante pour limiter les oscillations. Effectivement, sur CARLA les oscillations n'étaient pas forcément très impactantes du moment que l'agent réussissait à rester sur sa voie. Dans le véhicule réel, il est absolument nécessaire de limiter au maximum ces oscillations pour des raisons de confort.

Apprentissage distribué sur plusieurs machines Notre transformation d'image étant assez lente, nous avons dû utiliser plusieurs machines différentes afin de générer suffisamment d'expériences pour faire l'entraînement de l'agent par renforcement. Ainsi, nous avons utilisé cinq machines différentes, une utilisée pour l'algorithme d'apprentissage, les autres pour les 20 acteurs de l'entraînement.

7.4.3 Résultat et comparaison du renforcement par rapport à l'imitation

Nous avons entraîné nos différents agents sur une partie de notre base de données initiale (cf Chapitre 6 Section 6.2.2) représentant environ 30 heures et 1000km de conduite. Nous avons évalué notre agent appris par renforcement dans notre simulateur sur des enregistrements jamais vus représentant environ 10 heures et 250km de conduite. Comme dans le chapitre précédent, nous comptons le nombre de reprises en main en admettant qu'une reprise en main par le conducteur humain doit être exécutée si l'agent s'éloigne de plus d'un mètre de la trajectoire initiale. Nous avons ainsi une performance représentée

par la distance moyenne parcourue avant une intervention. Nous avons aussi calculé le taux d'autonomie (cf Chapitre 6 Équation 6.4) et l'écart moyen à la trajectoire optimale, comme introduit dans le chapitre précédent. Sur ces mêmes enregistrements, nous avons aussi testé notre agent appris par imitation entraîné lors de la première phase supervisé des indices implicites. Les résultats de ces expériences sont présentés sur la Table 7.1.

Agent évalué	Dist/Int	Autonomie	Écart moyen (MAD)
Imitation	2.1km	96.6%	14.4cm
Renforcement	10.4km	99.3%	9.5cm

TABLE 7.1 – Comparaison de la performance de notre agent appris par renforcement et de notre agent appris par imitation. Notre principale métrique est la distance moyenne parcourue avant une intervention. Nous avons aussi calculé le taux d'autonomie et l'écart moyen à la trajectoire optimale comme dans le Chapitre 6. Nous avons évalué les deux agents sur plus de 200km d'enregistrements jamais vus durant l'entraînement. Ces enregistrements contiennent des zones urbaines, des autoroutes, des rond-points et aussi de nombreux changements de voie.

Il est important de noter que nous ne pouvons pas directement comparer aux résultats du chapitre précédent, en particulier car maintenant nous n'enlevons plus les données où le conducteur humain est sur une intersection ou change de voie. Notre cas d'application est donc bien plus difficile que dans le chapitre précédent, et on constate bien une baisse des performances pour l'imitation (d'environ 99% à 96.6%). Cependant, les résultats de l'agent par renforcement sont bien meilleurs et sont comparables aux résultats du chapitre précédent alors que la tâche est bien plus complexe. Cela correspond bien aux résultats de Amini et al. [2020], l'apprentissage par renforcement permet d'améliorer les performances par rapport à l'apprentissage par imitation. Ce premier résultat est donc très encourageant et semble indiquer que le comportement serait très bon sur le vrai véhicule.

Cependant, comme l'évaluation n'a pas encore pu être faite directement sur le vrai véhicule, on pourrait se demander si l'agent appris par renforcement (ou même celui appris par imitation) n'a pas sur-appris sur les artefacts de l'image induits par notre transformation d'image. Pour se convaincre que ce n'était pas le cas, nous avons évalué nos agents en utilisant le simulateur sol/ciel, i.e. qui n'utilise pas une estimation de la profondeur pour faire la reprojection. Cette transformation induit elle-aussi des artefacts en particulier sur tous les objets en relief de la scène qui se retrouvent déformés (c.f. Figure 6.14 du chapitre précédent). Mais ces artefacts sont totalement différents de ceux qui apparaissent lorsque l'on fait la projection avec estimation de profondeur, les agents n'ont donc pas pu sur-apprendre sur ces autres artefacts. Les résultats de cette deuxième expérience sont présentés sur la Table 7.2.

Ces résultats sont particulièrement intéressants. On constate que changer totalement la projection utilisée n'a pas de véritable impact sur les performances, signifiant que les agents n'ont pas sur-appris sur les artefacts. Cela confirme aussi l'hypothèse émise en introduction de cette partie, que la projection avec une estimation de la distance ne

Agent évalué	Dist/Int	Autonomie	Écart moyen (MAD)
Imitation	1.91km	96.2%	15.1cm
Renforcement	12km	99.4%	9.4cm

TABLE 7.2 – Comparaison de la performance de notre agent appris par renforcement et de notre agent appris par imitation évalué en utilisant une projection avec l'hypothèse sol/ciel. L'idée est que les agents n'ont pas pu sur-apprendre à partir des artefacts issus de cette projection car ils ont été tous les deux entraînés avec une projection utilisant une estimation de la distance. Ce résultat est extrêmement prometteur, nos agents ont donc réellement appris à partir des informations pertinentes de l'image comme les lignes ou les trottoirs. De plus, notre agent appris par renforcement atteint des performances bien supérieures à celles de notre agent appris par imitation. Cela laisse penser que les performances sur véhicule réel seront bien meilleures que celles obtenues avec notre approche précédente (cf Chapitre 6).

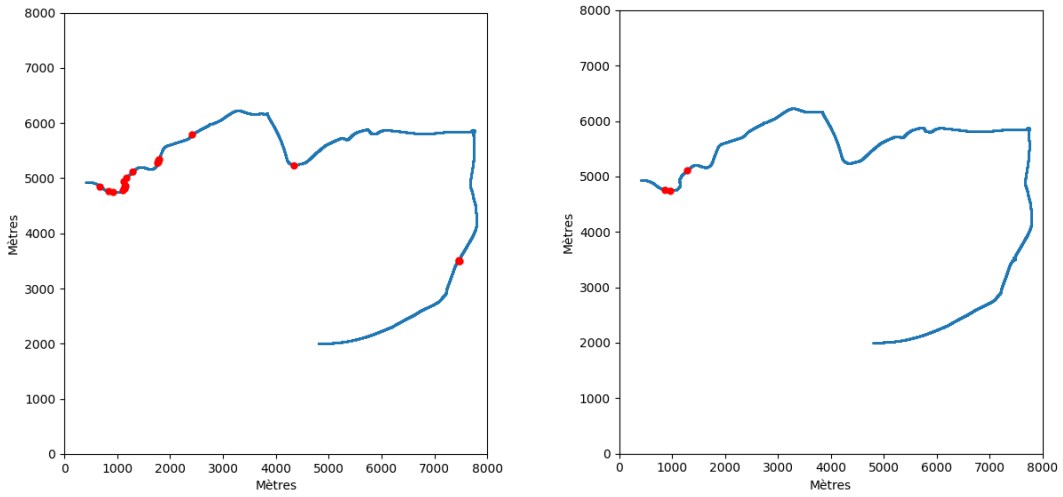
semble à priori pas réellement utile pour le contrôle latéral. En effet, les objets 3D de la scène n'ont pas un impact majeur sur le comportement latéral, la route et les marquages étant bien plus importants. Une visualisation des résultats de notre agent appris par renforcement comparé aux résultats de notre agent appris par imitation peut se trouver en Figure 7.12. On constate que l'agent appris par renforcement a un comportement meilleur nécessitant bien moins de reprises en main (chaque point rouge correspond à une reprise en main simulée) que l'agent appris par imitation.

Ces résultats sont extrêmement prometteurs, l'apprentissage par renforcement a des performances excellentes et bien supérieures à l'apprentissage par imitation et on a la preuve que ce n'est pas grâce aux artefacts de notre projection. Nous sommes ainsi convaincus que le contrôle latéral devrait fonctionner bien mieux que l'apprentissage par imitation précédent, même sur le véhicule réel. L'intégration sur le véhicule réel reste cependant la seule méthode pour véritablement le démontrer.

7.5 Renforcement pour le contrôle longitudinal

Après avoir présenté notre approche pour le contrôle latéral du véhicule et montré des résultats très prometteurs, nous détaillerons dans cette partie notre approche pour le contrôle longitudinal. Il est important de noter que ces travaux sont encore en cours de développement lors de l'écriture de ce manuscrit et que les résultats ne sont pas encore matures. Cette section est donc une description de ce qui a été fait actuellement, des premiers résultats et des points qu'il reste à développer et à améliorer dans de futurs travaux pour espérer faire fonctionner de l'apprentissage par renforcement pour le contrôle longitudinal.

Nous allons d'abord présenter un problème fondamental avec le simulateur fondé sur des images réelles lorsque l'on veut simuler des écarts en longitudinal : doit-on associer l'agent avec l'image la plus proche en distance (*association spatiale*) ou doit-on prendre l'image la plus proche en temps (*association temporelle*) ? Dans un deuxième temps, nous



(a) Agent appris par imitation

(b) Agent appris par renforcement

FIGURE 7.12 – Comparaison du nombre de reprises en main (les points rouges) entre notre agent appris par imitation (à gauche, 20 reprises en main) et notre agent appris par renforcement (à droite, 3 reprises en main). Les deux agents ont été évalués sur le même trajet d'environ 15km.

indiquerons quelles sont les indices nécessaires pour le contrôle longitudinal et pourquoi nous avons eu besoin de créer une nouvelle base de données avec ces indices. Ensuite, nous présenterons notre approche pour apprendre des indices implicites pertinents pour le contrôle longitudinal. En particulier, nous décrirons comment nous nous sommes inspirés de l'article de Hu et al. [2020] pour mieux gérer l'aspect temporel de l'entrée du réseau. En effet, cet aspect temporel semble bien plus important pour le contrôle longitudinal que pour le contrôle latéral, en particulier pour pouvoir déterminer la vitesse et les positions futures des objets dynamiques de la scène. Nous verrons aussi que la détection des états des feux tricolores est pour le moment très problématique et que cette tâche de perception est pour le moment bloquante à l'ensemble de cette partie. Finalement, nous décrirons notre configuration d'apprentissage par renforcement et nous présenterons de premiers résultats. Nous verrons ainsi que bien que l'agent arrive relativement bien à contrôler la vitesse sur les enregistrements vus durant l'entraînement, il généralise extrêmement mal sur des enregistrements jamais vus. En fait, nous avons pour le moment de grand problème de généralisation pour les différents indices, en particulier la détection de l'état des feux. C'est très probablement la raison pour laquelle nos agents appris par renforcement sont pour le moment incapables de généraliser à des données jamais vues par l'entraînement supervisé.

7.5.1 Association temporelle ou spatiale pour le simulateur

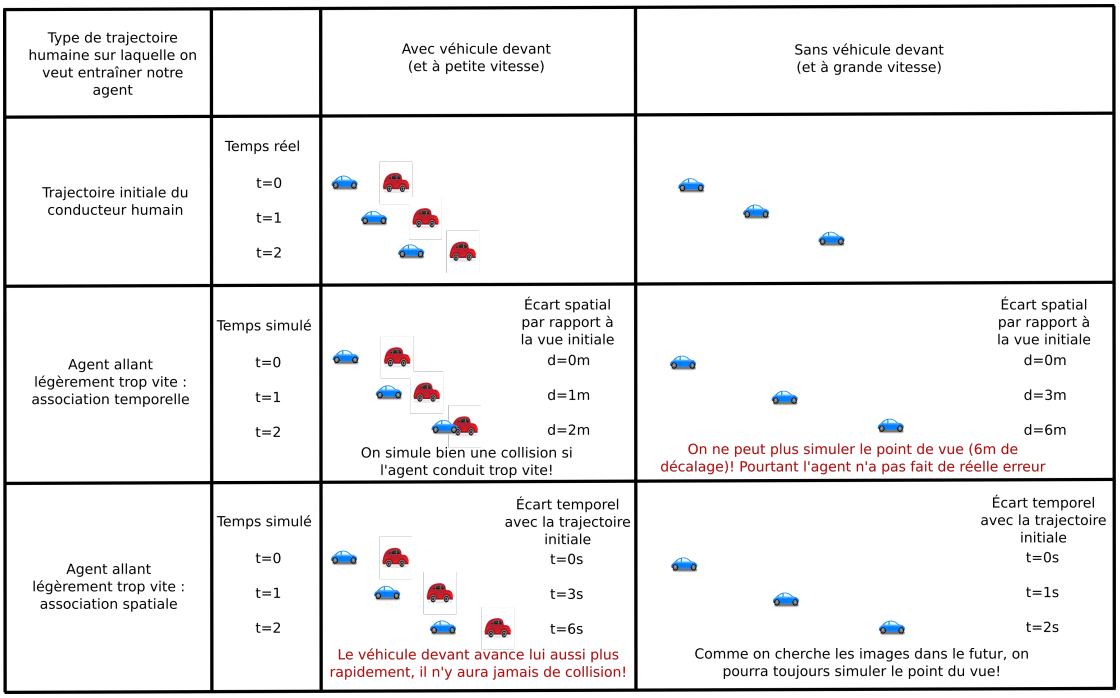


FIGURE 7.13 – Illustration de la différence entre association temporelle et association spatiale. A gauche, on a illustré un cas où le conducteur suit un autre véhicule à faible vitesse. Dans le cas idéal, on veut simuler une collision si l'agent conduit trop vite et se rapproche trop du véhicule précédent. On arrive bien à simuler cette collision si on effectue une association temporelle. Cependant, si on fait une association spatiale, tout se passera comme si le véhicule précédent accélère lui aussi. Dans cette situation, il est nécessaire d'effectuer une association temporelle pour que l'agent puisse apprendre à ne pas faire de collision. Cependant, si on prend la situation représentée à droite où le conducteur humain conduit rapidement sans aucun objet devant, l'association spatiale semble plus appropriée. En effet, si l'on fait une association temporelle et que l'agent conduit légèrement plus vite que le conducteur humain, alors l'agent va s'éloigner de plus en plus de la vraie position et on sera rapidement trop loin de la véritable position du véhicule pour pouvoir générer le point de vue. Pourtant l'agent n'aura pas réellement fait d'erreur car le même conducteur humain expert aurait tout aussi bien pu conduire à une vitesse légèrement différente dans cette situation. En fait, contrairement au contrôle latéral, il est possible de s'éloigner grandement de la position du conducteur humain sans avoir fait de réelles erreurs. Dans cette deuxième situation, l'association spatiale est meilleure car elle permet de pouvoir générer les points de vue sur une période beaucoup plus longue. Pour ces raisons, nous avons choisi d'utiliser une association temporelle seulement à faible vitesse et une association spatiale sinon.

La différence entre association temporelle et association spatiale est illustrée sur la Figure 7.13. L'association temporelle est celle que l'on a utilisée pour le contrôle latéral, on prend toujours l'image suivante dans la séquence pour simuler le point de vue suivant. Cette association a le grand avantage de préserver la consistance temporelle entre les images successives mais est limitée dès que l'agent se trouve trop loin du conducteur humain. L'association spatiale signifie qu'on génère le point de vue à partir de l'image la plus proche spatialement, cela permet en fait de générer un point de vue quelques soient les actions choisies par l'agent. Cependant, cette association détruit toute cohérence temporelle sur les objets dynamiques. En effet, si l'agent conduit deux fois plus vite et que l'on cherche les images dans le futur pour faire les projections, tous les objets mobiles donneront l'impression d'aller eux aussi deux fois plus vite. En fait, la différence fondamentale avec le latéral est qu'on ne peut pas facilement décider si une situation est mauvaise ou non après accumulation d'erreurs. En effet, si l'agent estime qu'il faut aller à 50km/h et que le conducteur humain est allé à 48km/h, on aimerait dire que l'agent conduit presque parfaitement, pourtant en accumulant les erreurs l'agent va se retrouver en moins de 15 secondes à plus de 10 mètres de la position réelle du conducteur humain et on ne pourra plus simuler le point de vue. Par contre, si il y a un véhicule devant allant à 48km/h, alors aller à 50km/h peut être extrêmement dangereux car l'agent va se rapprocher de plus en plus pour finalement faire une collision. On constate donc que l'accumulation d'erreurs est dans certains cas totalement anodine et dans d'autres extrêmement grave. Ce problème n'est pas présent pour le contrôle latéral, si l'agent est à plus d'un mètre sur le coté de la vraie position de l'humain cela signifie quasiment certainement que l'agent est sorti de sa voie et a fait de véritables erreurs.

Dans la grande majorité des cas, rouler à une vitesse légèrement différente du conducteur humain n'est pas une erreur car même le conducteur humain ne roulerait pas exactement à la même vitesse à chaque fois. C'est pourquoi l'association spatiale semble pertinente dans ces situations. En fait, les objets dynamiques sont la seule limitation de l'association spatiale, tous les objets fixes de la scène seront parfaitement cohérents. Ainsi, dans l'idéal on voudrait forcer une association temporelle si il y a un objet dynamique dans la scène courante et prendre une association spatiale sinon. C'est pour cela que nous avons décidé d'effectuer une association temporelle quand le conducteur humain est à une vitesse faible (inférieure à 10km/h) et une association spatiale dans tous les autres cas. En effet, lorsque le conducteur humain conduit à vitesse réduite c'est qu'il y a généralement des objets dynamiques dans la scène comme d'autres voitures ou des piétons.

7.5.2 Création d'une base de données pour les indices implicites

Pour notre travail sur CARLA présenté en Chapitre 5, la détection des feux tricolores était l'un des points les plus bloquants et nos expériences ont montré que cet indice était le plus important et permettait d'obtenir les meilleures performances lors de l'apprentissage par renforcement. Il semble donc logique que cette information soit aussi extrêmement importante pour le contrôle longitudinal en environnement urbain sur un véhicule réel. Cependant, nous n'avons qu'une seule base de données réelles où

les feux tricolores sont annotés et sa taille est assez réduite. De plus, l'ensemble des feux y sont annotés par une boîte avec leur état même ceux qui n'ont aucune interaction avec le conducteur humain. L'information que l'on voudrait vraiment avoir et est ce qu'il y a un feu s'appliquant à l'agent et si oui quelle est sa couleur. C'est pour cette raison que nous avons annoté manuellement des séquences pour obtenir une base de données avec exactement cette information. De plus, cela nous a aussi permis de retirer toutes les images où le conducteur humain s'arrête sans raison apparente et qui fausserait l'entraînement : soit l'agent n'arrivera pas à freiner et se trompera sans comprendre, soit il aura sur-appris sur la séquence d'entraînement. En fait annoter l'état des feux est bien plus simple et plus rapide qu'annoter les boites objets ou bien la segmentation sémantique, chaque image est associée avec un seul scalaire et il y a une cohérence temporelle, d'abord une succession d'images sans feu, puis des images avec feu rouge, ensuite le feu passe au vert pour finalement retourner sur des images sans feu. Ainsi, il nous aura suffit que d'une dizaine d'heures pour annoter plus de 100 000 images. Nous avons toujours utilisé cette base de données que ce soit pour entraîner nos indices implicites ou pour nos premières expériences d'apprentissage par renforcement.

7.5.3 Apprentissage d'indices implicites pour le contrôle longitudinal

L'apprentissage du contrôle longitudinal est beaucoup moins exploré que l'apprentissage du contrôle latéral. Cependant, il semble que l'information temporelle soit bien plus importante que pour le contrôle latéral, en particulier car il faut pouvoir détecter la vitesse des autres objets de la scène. De plus, une autre information qui semble pertinente est la présence ou non d'un objet devant le véhicule et la distance à cet objet. Finalement, comme pour notre travail sur CARLA, il semble nécessaire d'entraîner de manière supervisée à détecter les feux car cette tâche est extrêmement difficile et nous supposons qu'il est quasiment impossible de l'apprendre seulement à partir du signal d'apprentissage du renforcement.

Nous avons donc choisi d'utiliser de nombreux indices différents pour notre entraînement supervisé : la présence et l'état d'un feu tricolore, la présence et la distance au véhicule devant ainsi qu'une tâche d'imitation pour estimer la vitesse courante ainsi que l'accélération prise par le conducteur humain. L'ensemble de ces indices et l'architecture de notre encodeur-décodeur sont illustrés en Figure 7.14.

Il est important de noter que nous avons utilisé le *bloc temporel* introduit par Hu et al. [2020] pour donner des informations temporelles dès l'apprentissage supervisé. L'idée de ce bloc temporel est d'effectuer plusieurs convolutions 3D en parallèle avec soit des filtres sur la dimension temporelle soit sur la dimension spatiale. Cela s'inspire des convolutions en 2D en parallèle des réseaux supervisés standard comme ceux du réseau Inception [Szegedy et al., 2015]. Une illustration du bloc temporel de Hu et al. [2020] ainsi que du bloc du réseau Inception sont représentées sur la Figure 7.15.

Cependant, nous avons pour le moment de gros problèmes de sur-apprentissage lors de ce premier apprentissage supervisé. En particulier, la détection des feux a une précision de 100% sur la base d'entraînement alors que la sortie est quasiment aléatoire sur la base d'évaluation. La raison la plus probable à ce sur-apprentissage est que nous n'avons

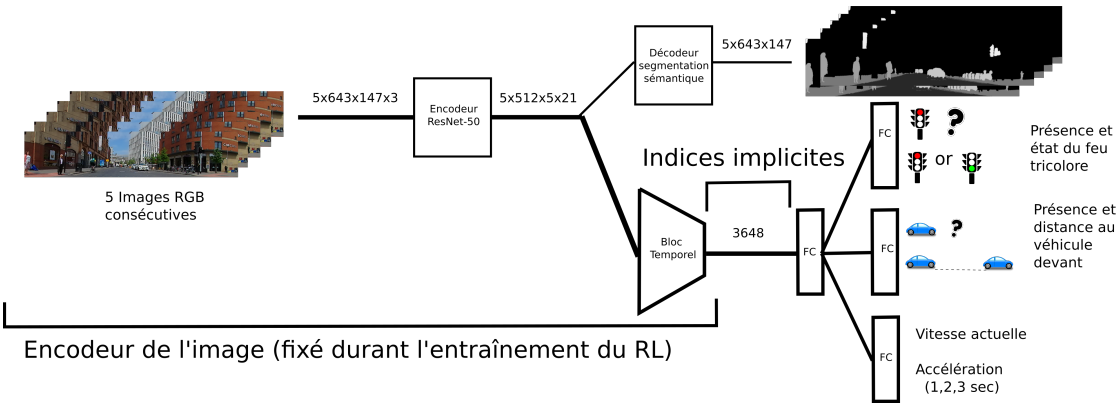
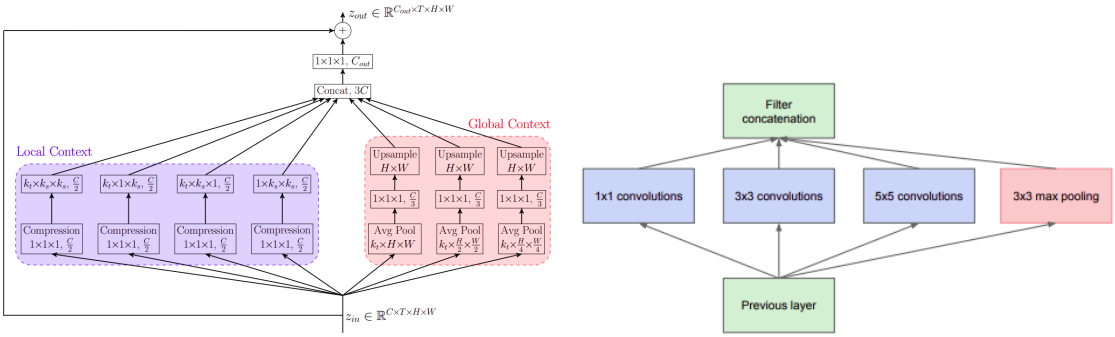


FIGURE 7.14 – Entraînement de nos indices implicites pour le contrôle longitudinal avec l'ensemble des fonctions de coût que l'on a utilisées. Nous avons choisi d'utiliser de multiples indices pour ce premier apprentissage. D'abord on estime une carte de segmentation pour chaque image d'entrée. Puis, après avoir appliqué un bloc temporel (cf Figure 7.15), nous entraînons le réseau à estimer d'autres informations, comme l'état du feu tricolore, la présence d'un véhicule devant, la vitesse courante et finalement la différence de vitesse (correspondant à une accélération) à 1, 2 et 3 secondes dans le futur. Cette dernière prédiction correspondant à l'imitation des accélérations prises par le conducteur humain et permettra ainsi d'avoir une référence d'apprentissage par imitation pour le contrôle longitudinal.



(a) Bloc temporel introduit par Hu et al. [2020] (b) Bloc Inception introduit par Szegedy et al. [2015]

FIGURE 7.15 – Illustration du bloc temporel introduit par Hu et al. [2020] qui prend en entrée des données avec quatre dimensions. On peut voir cette architecture comme une extension avec l'ajout d'une dimension temporelle au bloc Inception de Szegedy et al. [2015] qui prend des entrées en 3 dimensions.

pas assez de données pour bien généraliser et que la tâche de détection de feu est très difficile. En effet, la grande majorité de l'image n'a en fait aucun impact sur l'état du feu, et il semble probablement plus facile pour le réseau d'apprendre l'état du feu en fonction du contexte plutôt que d'apprendre à détecter les quelques pixels représentant réellement

l'état du feu. Il est important de noter que ces travaux sont encore peu matures et qu'il est donc trop tôt pour savoir précisément quelle est la raison de cet échec.

7.5.4 Configuration de l'apprentissage par renforcement

Même si les premiers apprentissages supervisés ne généralisent pas à la base de données d'évaluation, les performances sur la base d'entraînement sont parfaites et cela nous semblait une bonne première expérience d'apprendre par renforcement le contrôle longitudinal sur cette base d'entraînement. Cela permet en effet de voir si il y a des problèmes potentiels et si l'agent réussit bien à apprendre à conduire lorsque les indices implicites sont parfaits.

Dans cette section, nous allons présenter la configuration de notre apprentissage par renforcement, en particulier quel est l'espace d'états et d'actions que l'on a choisi et comment nous avons construit notre fonction de récompense.

Espace d'états De même que pour l'apprentissage par renforcement pour le contrôle latéral, nous avons choisi de concaténer plusieurs indices implicites consécutifs et d'utiliser cet ensemble comme état pour notre agent. Ainsi, l'aspect temporel peut être pris en compte à la fois lors du premier apprentissage grâce à l'utilisation du bloc temporel [Hu et al., 2020] et aussi durant l'apprentissage par renforcement car l'agent peut accéder aux indices implicites précédents. Nous avons choisi cela car nous supposons que pour le contrôle longitudinal, l'aspect temporel est primordial en particulier pour détecter les objets mobiles, leurs directions et leurs vitesses.

Espace d'actions Nous avons aussi utilisé 57 actions différentes, chaque action étant la dérivée de la vitesse, i.e. l'accélération, que l'on veut appliquer. Nous avons limité l'accélération et le freinage maximaux en utilisant les valeurs utilisées par les algorithmes de contrôle classique déjà présents dans les véhicules Valeo. Ainsi, l'accélération maximale est limitée à 1.5m/s^2 (0.15G) et le freinage maximal à 3m/s^2 (0.3G). Les actions sont uniformément réparties sur cet intervalle.

Fonction de récompense Notre fonction de récompense varie grandement en fonction de si l'on utilise une association spatiale ou une association temporelle, c.f. section 7.5.1. En effet, lorsque l'on utilise une association temporelle, l'agent va accumuler ses erreurs d'une image à l'autre et on utilise donc une récompense simple qui dépend de la distance entre l'agent et le conducteur humain. Cette récompense est maximale égale à 1 lorsque l'agent est exactement à la position du conducteur (et au même instant) et descend jusqu'à zéro si l'agent s'éloigne à plus de 3 mètres du conducteur. Il est en effet difficile de simuler des points de vue trop éloignés de la véritable position du conducteur humain, plus on s'éloigne plus il y a des artefacts et des pixels vides dans le point de vue généré. C'est pour cette raison que nous permettons à l'agent seulement 3 mètres de décalage avec le conducteur humain. Nous rappelons que nous n'utilisons une association temporelle que quand la vitesse du conducteur humain est faible ainsi 3 mètres

de décalage est généralement une vraie erreur de l'agent : soit il a freiné beaucoup trop tôt, soit il est quasiment rentré dans l'obstacle situé au devant.

Par contre, lorsque l'on utilise une association spatiale, on ne peut plus utiliser la récompense présentée précédemment dépendant de la différence de position entre l'agent et le conducteur humain. Ainsi, nous utilisons dans ce cas une récompense qui dépend de l'écart entre la vitesse de l'agent et la vitesse du conducteur humain.

En outre, nous utilisons dans tous les cas une récompense de confort, i.e cette récompense est maximale lorsque l'agent garde la même vitesse et diminue lorsque l'accélération calculée par l'agent augmente. Cette récompense devrait permettre d'éviter que la vitesse de l'agent oscille trop. Comme pour les oscillations latérales, dans le simulateur CARLA, avoir une vitesse qui change rapidement n'a finalement qu'assez peu d'impact sur les performances mais cela n'est évidemment pas possible dans un vrai véhicule. Finalement, si la vitesse de l'agent est trop éloignée de celle du conducteur humain (nous avons pris un seuil à 10km/h), l'agent reçoit une récompense de -1 et l'épisode se termine.

7.5.5 Résultat préliminaire de l'apprentissage par renforcement

Nous avons entraîné nos différents agents sur les données où l'on a annoté manuellement la présence et l'état d'un feu tricolore pertinent pour l'agent, cf Section 7.5.3. Cette base de données représente environ 5 heures et 150km de conduite. Cela représente plus de six fois moins de données que celles utilisées pour l'apprentissage du contrôle latéral et semble une raison probable au manque de généralisation de nos différents agents.

Pour évaluer nos agents, nous comptons le nombre de reprises en main en admettant qu'une reprise en main est nécessaire si la vitesse de l'agent est trop éloignée de la vitesse du conducteur humain. Nous comparons ensuite nos agents à une référence qui conduit à vitesse constante. Cette référence semble extrêmement naïve mais contrairement au contrôle latéral, il n'y a pas réellement de référence déjà existante avec des performances raisonnables et nous n'avons pas eu le temps de définir une véritable métrique pour évaluer plus proprement nos agents. Les résultats de ces expériences sont présentés sur la Table 7.3.

Ainsi, nous avons remarqué que nos agents appris par renforcement ont un comportement raisonnable sur les trajectoires vues durant l'entraînement dans le sens où ils nécessitent beaucoup moins de reprises en main que la référence naïve (cf Table 7.3). Nous avons aussi observé qualitativement que l'agent est capable de s'arrêter aux feux rouges ou lorsqu'il y a un véhicule devant sur les enregistrements utilisés pour l'entraînement.

Cependant, les performances de notre agent appris par renforcement sont très mauvaises sur les enregistrements gardés pour l'évaluation : quasiment identiques à un agent conduisant à une vitesse constante. Cela était attendu car pour le moment nos indices implicites (l'état du feu ou la présence d'un obstacle par exemple) ne généralisent pas à des données jamais vues, ainsi les features utilisées comme état pour notre agent ne contiennent pas les informations nécessaires à la conduite. Il reste donc encore beaucoup de travail avant de réellement obtenir un agent appris par renforcement capable de généralisation pour le contrôle longitudinal.

Train/Test	Train		Test	
Métrique	Dist/Int (m)	Aut. (%)	Dist/Int (m)	Aut. (%)
Imitation	360m	85%	210m	74%
RL	1000m	95%	210m	73%
Référence	150m	66%	160m	68%

TABLE 7.3 – Comparaison de la performance de notre agent appris par renforcement et de notre agent appris par imitation avec une référence naïve qui conduit à vitesse constante (vitesse initiale du conducteur humain). Nos deux métriques principales sont la distance moyenne parcourue avant une reprise en main ainsi que le pourcentage d'autonomie. On remarque que sur les trajectoires vues durant l'entraînement, l'imitation et surtout le RL ont de bien meilleures performances que la référence naïve : plus de six fois moins de reprises en main pour l'agent appris par renforcement. Par contre, sur les trajectoires d'évaluation, les performances de l'imitation et du RL sont très mauvaises : quasiment les mêmes que la référence naïve qui conduit à vitesse constante. Cela signifie que nos différents agents ne sont pas encore capables de généraliser à d'autres données.

7.6 Conclusion

Dans cette section, nous avons présenté les derniers travaux de cette thèse cherchant à appliquer du renforcement pour le contrôle latéral et longitudinal d'un véhicule réel en environnement urbain. Nos premiers résultats sont extrêmement encourageants pour le contrôle latéral et semblent prometteurs pour une intégration rapide dans un vrai véhicule. Pour cela nous avons combiné les idées de différents travaux, tout d'abord les indices implicites provenant de notre propre article [Toromanoff et al., 2020], puis l'idée de reprojection utilisant une estimation de la profondeur comme introduit par Amini et al. [2020] et finalement notre propre travail d'apprentissage par imitation [Toromanoff et al., 2018] sur un véhicule réel pour l'augmentation de labels ainsi que l'ensemble des outils concernant les données réelles. Notre agent final appris par renforcement atteint des performances bien meilleures que notre agent appris par imitation. De plus, nous avons même certifié que l'agent ne sur-apprenait pas des artefacts sur l'image qui surviennent lors de nos augmentations de point de vues car nous l'avons évalué avec une projection totalement différente sans que cela n'ait d'impact majeur sur les performances.

Par contre, le contrôle longitudinal n'est pas encore assez mature et de nombreux développements sont encore nécessaires, en particulier concernant la détection des feux tricolores, avant de pouvoir espérer obtenir des résultats raisonnables sur un véhicule réel. Cela est en fait tout à fait logique, la grande majorité des travaux d'apprentissage du contrôle sur véhicule réel, que ce soit par renforcement ou par imitation, s'appliquent seulement au contrôle latéral. Cela signifie qu'il y a un grand fossé entre ces deux tâches, et nous sommes convaincus que notre approche propose des idées intéressantes qui permettront d'obtenir, potentiellement pour la première fois, des résultats sur un vrai véhicule pour le contrôle longitudinal par renforcement. Ceci est donc laissé en travaux

futurs de cette thèse, et des idées d'amélioration seront détaillées en conclusion finale de cette thèse.

Chapitre 8

Conclusion

8.1 Contributions

Dans cette thèse, nous avons présenté des approches incrémentales dont le but final est de réussir à appliquer de l'apprentissage de bout-en-bout par renforcement pour la conduite d'un véhicule réel en environnement urbain. Cela a en particulier été possible grâce à l'élaboration d'un nouvel algorithme de renforcement distribué (cf Chapitre 4) ainsi que la technique des *indices implicites* introduite dans le Chapitre 5.

Dans un premier temps, nous avons proposé un nouvel algorithme de renforcement fondé sur la fonction de valeur, Rainbow-IQN Ape-X, en combinant trois articles majeurs du domaine. Cet algorithme atteint des performances au niveau de l'état de l'art sur le benchmark Atari. Afin de permettre de mieux reproduire les résultats, nous avons défini un nouveau standard d'évaluation sur le benchmark Atari ainsi que fourni une implémention open-source de notre code. L'aspect parallélisable de notre algorithme est fondamental pour l'application à des domaines où la génération de données n'est pas aussi rapide que sur les jeux Atari. En effet, nous n'aurions pas pu faire converger nos apprentissages sur le simulateur CARLA [Dosovitskiy et al., 2017] ou sur les données réelles en un temps raisonnable sans notre architecture distribuée.

En utilisant cet algorithme de renforcement distribué, nous avons introduit les *indices implicites*, une nouvelle méthode permettant d'entraîner par renforcement des réseaux de neurones avec bien plus de paramètres et des entrées de plus grande dimension que les travaux précédents en DRL. L'idée principale est de décomposer l'entraînement en deux phases. Tout d'abord, une phase d'entraînement supervisé de différents indices utiles à la conduite comme l'estimation d'une carte de segmentation ou de l'état du feu tricolore. Dans une deuxième phase, on utilise les features provenant du réseau pré-entraîné, qui est fixé durant l'entraînement par renforcement, comme entrée pour notre algorithme de renforcement. Nous avons nommé cette technique *indices implicites* car l'agent n'utilise pas les indices explicites mais seulement les features qui permettent à notre premier réseau d'estimer les différents indices. Cette technique nous a ainsi permis de démontrer pour la première fois un algorithme de renforcement capable de conduire dans un simulateur complexe incluant des piétons, des véhicules et surtout des feux tricolores. Nous avons aussi remporté deux fois le CARLA Challenge (en 2019 et en 2020) sur la catégorie "Cameras Only".

Dans un travail parallèle, nous avons proposé une méthode permettant de générer automatiquement des situations d'échec et leurs corrections associées pour le contrôle latéral en utilisant une unique caméra frontale fisheye Valeo. Cela diffère des méthodes préexistantes car nous n'avons besoin que d'une seule caméra et non pas de plusieurs

caméras pour générer ces situations d'échec. Nous avons ensuite entraîné un agent par imitation en incorporant ces situations d'échec et leurs corrections associées dans la base d'entraînement. Ces augmentations permettent à l'agent de réagir à ses propres erreurs lorsqu'il est évalué en boucle fermé. Finalement, nous avons intégré ce travail dans un véritable véhicule pour l'évaluer en route ouverte ainsi que pour une démonstration au CES 2019 à Las Vegas.

Enfin, nous avons utilisé toutes nos contributions précédentes pour effectuer de l'apprentissage par renforcement sur données réelles pour de la conduite en environnement urbain. En effet, nous avons d'abord entraîné un réseau à estimer différents indices comme la segmentation sémantique ou l'état du feu tricolore sur données réelles. Nous avons aussi entraîné ce même réseau par imitation en utilisant les augmentations mentionnées précédemment, l'idée étant que l'imitation est en fait un indice de conduite particulièrement pertinent. Finalement, nous avons utilisé notre algorithme de renforcement distribué en utilisant les indices implicites issus de ce premier réseau comme état pour nos agents appris par renforcement. Nos résultats pour le contrôle latéral sont particulièrement prometteurs même si l'intégration véritable sur le véhicule reste à faire. Concernant le contrôle longitudinal, les résultats sont encore immatures et plus de développements sont nécessaires avant de pouvoir réellement intégrer nos modèles dans le vrai véhicule. Il est important de noter qu'il n'y a actuellement aucun article publié ayant intégré de l'apprentissage par renforcement sur un vrai véhicule pour conduire en milieu urbain. Nos très bons résultats, pour le contrôle latéral au moins, sont donc très intéressants et vont être rapidement publiés dans les travaux futurs de cette thèse.

8.2 Publications

Cette thèse a abouti aux publications suivantes :

— Perot, E., Jaritz, M., **Toromanoff, M.**, and de Charette, R.
End-to-end driving in a realistic racing game with deep reinforcement learning.
CVPR Workshop 2017.

— Jaritz, M., de Charette, R., **Toromanoff, M.**, Perot, E., and Nashashibi, F.
End-to-end race driving with deep reinforcement learning. *ICRA 2018.*

— **Toromanoff, M.**, Wirbel, E., Wilhelm, F., Vejarano C., Perrotton, X., and Moutarde, F.
End to end vehicle lateral control using a single fisheye camera. *IROS 2018.*

— **Toromanoff, M.**, Wirbel, E., and Moutarde, F.
Is Deep Reinforcement Learning Really Superhuman on Atari ?
NeurIPS Workshop 2019.

— **Toromanoff, M.**, Wirbel, E., and Moutarde, F.
End-to-end model-free reinforcement learning for urban driving using implicit affordances. *CVPR 2020.*

8.3 Travaux futurs

Dans le dernier travail de cette thèse, nous avons cherché à appliquer de l'apprentissage par renforcement de bout-en-bout pour la conduite en environnement urbain en utilisant un simulateur fondé sur des images réelles comme celui introduit par Amini et al. [2020].

Cependant, comme nous l'avons vu, nous avons des problèmes de sur-apprentissage pour la détection du feu tricolore pertinent qui semble le point le plus bloquant au contrôle longitudinal actuellement. Ce problème de pure perception semble aujourd'hui assez peu exploré : la majorité des articles détectent en fait tous les feux, qu'ils soient pertinents ou non [Shi et al., 2015, Behrendt et al., 2017, Lee and Kim, 2019]. D'autres travaux [Possatti et al., 2019] reposent sur une carte HD pour détecter le feu pertinent dans l'image mais cela nécessite donc de parcourir en amont la route que l'on veut suivre pour indiquer la position des différents feux tricolores pertinents. Détecter le feu pertinent à partir seulement d'images caméra est donc probablement un des travaux futurs les plus importants à effectuer.

Un autre travail futur pertinent serait d'améliorer la prédiction de profondeur. En effet, comme indiqué dans le chapitre précédent, des petites erreurs de distance peuvent générer des artefacts visuels très importants surtout en cas de zooms conséquents. On pourrait donc utiliser des algorithmes de prédiction de profondeur auto-supervisés plus performants ou alors utiliser des annotations provenant d'un LIDAR.

Enfin, améliorer notre génération de point de vue en utilisant les travaux de l'état de l'art en *affichage de points 3D sur des images* (en anglais *image-based rendering of point cloud*) permettrait d'améliorer notre simulateur fondé sur image réelle.

Concernant des problèmes plus globaux et plus long terme, on peut penser à la gestion de la temporalité. Concaténer des images consécutives ou même le bloc temporel introduit par Hu et al. [2020] permet seulement de prendre en compte un nombre statique d'images et ces approches sont donc nécessairement très limitées dans le temps. L'utilisation de réseau de neurones récurrents comme les LSTM [Hochreiter and Schmidhuber, 1997] pourrait permettre une mémoire variable et pouvant utiliser des informations plus loin dans le passé. Cependant, entraîner des réseaux récurrents, en particulier avec des entrées de très grande dimension, est encore un travail très difficile.

Pour finir, même si toute notre architecture atteint finalement de très bons résultats, il restera un problème inhérent à l'utilisation d'un simulateur fondé sur des trajectoires réelles. En effet, comme nous utilisons des trajectoires pré-enregistrées, quelles que soit les actions de l'agent, les autres usagers de la route ne vont jamais modifier leur comportement : l'agent est comme un fantôme naviguant dans un monde ignorant son existence. Je reste convaincu que ce type de simulateur peut permettre d'apprendre un comportement dans des situations particulièrement diverses mais les interactions très fines entre l'agent et les autres participants peuvent potentiellement être annihilées par ce problème de *l'agent fantôme*. Un apprentissage final sur la voiture réelle comme utilisé par Kendall et al. [2019], i.e. en utilisant les reprises en main du conducteur comme signal d'apprentissage, serait donc probablement nécessaire pour gérer ces situations particulièrement

complexes.

8.4 Messages clés de cette thèse

Au lancement de cette thèse en janvier 2018, nous voulions étudier l'intérêt de l'apprentissage par renforcement qui n'était alors, et n'est toujours d'ailleurs, que très peu utilisé dans le cadre de la voiture autonome. Pour réussir à appliquer du renforcement à la conduite autonome, nous avons d'abord introduit un nouvel algorithme de renforcement distribué (cf Chapitre 4) permettant d'utiliser de nombreuses ressources de calcul en parallèle et ainsi de diminuer nettement le temps global de nos apprentissages. En effet, l'un des plus grands obstacles de l'apprentissage par renforcement est qu'il nécessite généralement une énorme quantité de données pour converger. En utilisant cet algorithme distribué et notre technique des *indices implicites* (cf Chapitre 5), nous avons démontré pour la première fois un agent appris par renforcement capable de naviguer dans un environnement urbain simulé complexe incluant d'autres véhicules, des piétons ainsi que des feux tricolores. Cette approche nous a en particulier permis de remporter deux fois le Challenge CARLA en 2019 et en 2020. Finalement, nous avons introduit un simulateur fondé sur des images réelles pour transférer notre approche sur des données réelles, obtenant ainsi des résultats très prometteurs pour le contrôle latéral d'un véhicule en environnement urbain réel.

Nous pensons que cette thèse a permis de mettre en lumière les obstacles principaux à l'application du renforcement à la voiture autonome, a introduit différentes méthodes pour pallier ces problèmes, et finalement apporte des axes de recherche prometteurs pour véritablement intégrer un apprentissage par renforcement sur un véhicule réel.

Bibliographie

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow : Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv :1603.04467*, 2016.

Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv :1910.07113*, 2019.

Berat Mert Albaba and Yildiray Yildiz. Driver modeling through deep reinforcement learning and behavioral game theory. *arXiv preprint arXiv :2003.11071*, 2020.

Pablo Aldape and Samuel Sowell. Reinforcement Learning for a Simple Racing Game, 2018.

Alexander Amini, Igor Gilitschenski, Jacob Phillips, Julia Moseyko, Rohan Banerjee, Sertac Karaman, and Daniela Rus. Learning robust control policies for end-to-end autonomous driving from data-driven simulation. *IEEE Robotics and Automation Letters*, 5(2) :1143–1150, 2020.

Zhengwei Bai, Wei Shangguan, Baigen Cai, and Linguo Chai. Deep reinforcement learning based high-level driving behavior decision-making model in heterogeneous traffic. In *2019 Chinese Control Conference (CCC)*, pages 8600–8605. IEEE, 2019.

Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet : Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv :1812.03079*, 2018.

Karsten Behrendt, Libor Novak, and Rami Botros. A deep learning approach to traffic lights : Detection, tracking, and classification. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1370–1377. IEEE, 2017.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment : An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47 :253–279, 2013.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

Alex Bewley, Jessica Rigley, Yuxuan Liu, Jeffrey Hawke, Richard Shen, Vinh-Dieu Lam, and Alex Kendall. Learning to drive from simulation without real world labels. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 4818–4824. IEEE, 2019.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv :1604.07316*, 2016.

Hassan Bouchiba, Jean-Emmanuel Deschaud, and Francois Goulette. Raw point cloud deferred shading through screen space pyramidal operators. In *39th Eurographics*, 2018.

Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G Bellemare. Dopamine : A research framework for deep reinforcement learning. *arXiv preprint arXiv :1812.06110*, 2018.

Yuan Chang and WANG Guo-Ping. A review on image-based rendering. *Virtual Reality & Intelligent Hardware*, 1(1) :39–54, 2019.

Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving : Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE international conference on computer vision*, pages 2722–2730, 2015.

Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR, 2020a.

Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. Model-free deep reinforcement learning for urban autonomous driving. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2765–2771. IEEE, 2019.

Jianyu Chen, Shengbo Eben Li, and Masayoshi Tomizuka. Interpretable end-to-end urban autonomous driving with latent deep reinforcement learning. *arXiv preprint arXiv :2001.08726*, 2020b.

Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4693–4700. IEEE, 2018.

Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9329–9338, 2019.

Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Proceedings of the 5th International Conference on Computer and Games*, volume 4630, 05 2006.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*, pages 1096–1105. PMLR, 2018.

Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

Pim de Haan, Dinesh Jayaraman, and Sergey Levine. Causal confusion in imitation learning. *NeurIPS*, 2019.

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla : An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-explore : a new approach for hard-exploration problems. *arXiv preprint arXiv :1901.10995*, 2019.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala : Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

Gilles Tagne Fokam. *Commande et planification de trajectoires pour la navigation de véhicules autonomes*. PhD thesis, Université de Technologie de Compiègne, 2014.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv :1706.10295*, 2017.

Laura García Cuenca, Enrique Puertas, Javier Fernandez Andres, and Nourdine Aliane. Autonomous driving in roundabout maneuvers using reinforcement learning with q-learning. *Electronics*, 8(12) :1536, 2019.

Laurent George, Thibault Buhet, Emilie Wirbel, Gaetan Le-Gall, and Xavier Perrotton. Imitation learning for end to end vehicle longitudinal control with forward camera. *arXiv preprint arXiv :1812.05841*, 2018.

Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 270–279, 2017.

Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv :1406.2661*, 2014.

Daniel Gordon, Abhishek Kadian, Devi Parikh, Judy Hoffman, and Dhruv Batra. Splitnet : Sim2sim and task2task transfer for embodied visual navigation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1022–1031, 2019.

Fabio M. Graetz. How to match DeepMind's Deep Q-Learning score in Breakout. https://tinyurl.com/yymceamo, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic : Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

Jeffrey Hawke, Richard Shen, Corina Gurau, Siddharth Sharma, Daniele Reda, Nikolay Nikolov, Przemysław Mazur, Sean Micklethwaite, Nicolas Griffiths, Amar Shah, et al. Urban driving with conditional imitation learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 251–257. IEEE, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow : Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. Darla : Improving zero-shot transfer in reinforcement learning. In *International Conference on Machine Learning*, pages 1480–1490. PMLR, 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8) :1735–1780, 1997.

Gabriel M Hoffmann, Claire J Tomlin, Michael Montemerlo, and Sebastian Thrun. Autonomous automobile trajectory tracking for off-road driving : Controller design, experimental validation and racing. In *2007 American control conference*, pages 2296–2301. IEEE, 2007.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv :1803.00933*, 2018.

Anthony Hu, Fergal Cotter, Nikhil Mohan, Corina Gurau, and Alex Kendall. Probabilistic future prediction for video scene understanding. In *European Conference on Computer Vision*, pages 767–785. Springer, 2020.

Christian Hubschneider, Andre Bauer, Michael Weber, and J Marius Zöllner. Adding navigation to the equation : Turning decisions for end-to-end vehicle control. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–8. IEEE, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization : Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv :1611.05397*, 2016.

Maximilian Jaritz, Raoul De Charette, Marin Toromanoff, Etienne Perot, and Fawzi Nashashibi. End-to-end race driving with deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2070–2075. IEEE, 2018.

Kaixhin. Open source implementation of rainbow by kaixhin, 2018. URL `https://github.com/Kaixhin/Rainbow`.

Danial Kamran, Carlos Fernandez Lopez, Martin Lauer, and Christoph Stiller. Risk-aware high-level decisions for automated driving at occluded intersections with reinforcement learning. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1205–1212. IEEE, 2020.

Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

Sagi Katz and Ayellet Tal. On the visibility of point clouds. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1350–1358, 2015.

Sagi Katz, Ayellet Tal, and Ronen Basri. Direct visibility of point sets. In *Proceedings of the ACM Transactions on Graphics*, volume 26, 07 2007. doi: 10.1145/1275808.1276407.

Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8248–8254. IEEE, 2019.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv :1312.6114*, 2013.

Charvak Kondapalli, Debraj Roy, and Nishan Srishankar. Deep Reinforcement Learning Applied to a Racing Game, 2017.

Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1094–1099. IEEE, 2015.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25 :1097–1105, 2012.

Varun Ravi Kumar, Sandesh Athni Hiremath, Markus Bach, Stefan Milz, Christian Witt, Clément Pinard, Senthil Yogamani, and Patrick Mäder. Fisheyedistancenet : Self-supervised scale-aware distance estimation using monocular fisheye camera for autonomous driving. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 574–581. IEEE, 2020.

Eunseop Lee and Daijin Kim. Accurate traffic light detection using deep neural network with focal regression loss. *Image and Vision Computing*, 87 :24–36, 2019.

Edouard Leurent. A survey of state-action representations for autonomous driving. *hal preprint hal01908175*, 2018.

Sergey Levine. CS 294 : Deep Reinforcement Learning. Technical report, UC Berkeley, 2017.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv :1509.02971*, 2015.

Marion Luyat and Tony Regia-Corte. Les affordances : de james jerome gibson aux formalisations récentes du concept. *LAnnee psychologique*, 109(2) :297–332, 2009.

Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment : Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61 :523–562, 2018.

Mark Martinez, Chawin Sitawarin, Kevin Finch, Lennart Meincke, Alex Yablonski, and Alain Kornhauser. Beyond grand theft auto v for training, testing and enhancing deep learning in self driving cars. *arXiv preprint arXiv :1712.01397*, 2017.

Ashish Mehta, Adithya Subramanian, and Anbumani Subramanian. Learning end-to-end autonomous driving using guided auxiliary supervision. In *Proceedings of the 11th Indian Conference on Computer Vision, Graphics and Image Processing*, pages 1–8, 2018.

Branka Mirchevska, Christian Pek, Moritz Werling, Matthias Althoff, and Joschka Boedecker. High-level decision making for safe and reasonable autonomous lane changing using reinforcement learning. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2156–2162. IEEE, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540) : 529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

Fabien Moutarde, Alexandre Bargeton, Anne Herbin, and Lowik Chanussot. Robust on-vehicle real-time visual detection of american and european speed limit signs, with a modular traffic signs recognition system. In *2007 IEEE Intelligent Vehicles Symposium*, pages 1122–1126. IEEE, 2007.

Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

OpenAI. OpenAI Five. https://openai.com/five/, 2018.

Błażej Osiński, Adam Jakubowski, Paweł Zięcina, Piotr Miłoś, Christopher Galias, Silviu Homoceanu, and Henryk Michalewski. Simulation-based reinforcement learning for real-world autonomous driving. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6411–6418. IEEE, 2020.

Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. Count-based exploration with neural density models. In *International conference on machine learning*, pages 2721–2730. PMLR, 2017.

Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv :1704.03952*, 2017.

Xinlei Pan, Xiangyu Chen, Qizhi Cai, John Canny, and Fisher Yu. Semantic predictive control for explainable and efficient policy learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3203–3209. IEEE, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch : An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors,

Advances in Neural Information Processing Systems 32, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, pages 2778–2787. PMLR, 2017.

Etienne Perot, Maximilian Jaritz, Marin Toromanoff, and Raoul De Charette. End-to-end driving in a realistic racing game with deep reinforcement learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 3–4, 2017.

J. Pineau. Reproducibility checklist. https://www.cs.mcgill.ca/~jpineau/ReproducibilityChecklist.pdf, 2019.

Tobias Pohlen, Bilal Piot, Todd Hester, Mohammad Gheshlaghi Azar, Dan Horgan, David Budden, Gabriel Barth-Maron, Hado Van Hasselt, John Quan, Mel Večerík, et al. Observe and look further : Achieving consistent performance on atari. *arXiv preprint arXiv :1805.11593*, 2018.

Dean A Pomerleau. Alvinn : An autonomous land vehicle in a neural network. Technical report, Carnegie-Mellon Univ Pittsburgh PA Artificial Intelligence and Psychology, 1989.

Lucas C Possatti, Rânik Guidolini, Vinicius B Cardoso, Rodrigo F Berriel, Thiago M Paixão, Claudine Badue, Alberto F De Souza, and Thiago Oliveira-Santos. Traffic light recognition using deep learning and prior maps for autonomous cars. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet : Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

Xiangjun Qian, Jean Gregoire, Fabien Moutarde, and Arnaud De La Fortelle. Priority-based coordination of autonomous and legacy vehicles at intersection. In *17th international IEEE conference on intelligent transportation systems (ITSC)*, pages 1166–1171. IEEE, 2014.

Xiangjun Qian, Jean Gregoire, Arnaud De La Fortelle, and Fabien Moutarde. Decentralized model predictive control for smooth coordination of automated vehicles at intersection. In *2015 European control conference (ECC)*, pages 3452–3458. IEEE, 2015.

Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data : Ground truth from computer games. In *European conference on computer vision*, pages 102–118. Springer, 2016.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net : Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

G. Rummery and Mahesan Niranjan. On-line q-learning using connectionist systems. *Technical Report CUED/F-INFENG/TR 166*, 11 1994.

Axel Sauer, Nikolay Savinov, and Andreas Geiger. Conditional affordance learning for driving in urban environments. In *Conference on Robot Learning*, pages 237–252. PMLR, 2018.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv :1511.05952*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv :1707.06347*, 2017.

Loren Arthur Schwarz. Non-rigid registration using free-form deformations. *Technische Universität München*, 6, 2007.

Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim : High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.

Evan Shelhamer, Parsa Mahmoudieh, Max Argus, and Trevor Darrell. Loss is its own reward : Self-supervision for reinforcement learning. *arXiv preprint arXiv :1612.07307*, 2016.

Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883, 2016.

Zhenwei Shi, Zhengxia Zou, and Changshui Zhang. Real-time traffic light detection with adaptive background suppression filter. *IEEE Transactions on Intelligent Transportation Systems*, 17(3) :690–700, 2015.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587) :484–489, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676) :354–359, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv :1409.1556*, 2014.

Jarrod M Snider et al. Automatic steering methods for autonomous automobile path tracking. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08*, 2009.

Richard S Sutton and Andrew G Barto. *Reinforcement learning : An introduction*. MIT press, 2018.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

E. Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306 vol. 1, 2005.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco : A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

Marin Toromanoff, Emilie Wirbel, Frédéric Wilhelm, Camilo Vejarano, Xavier Perrotton, and Fabien Moutarde. End to end vehicle lateral control using a single fisheye camera. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3613–3619. IEEE, 2018.

Marin Toromanoff, Emilie Wirbel, and Fabien Moutarde. Is deep reinforcement learning really superhuman on atari ? In *Deep Reinforcement Learning Workshop of 39th Conference on Neural Information Processing Systems (Neurips' 2019)*, 2019.

Marin Toromanoff, Emilie Wirbel, and Fabien Moutarde. End-to-end model-free reinforcement learning for urban driving using implicit affordances. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7153–7162, 2020.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Hado P van Hasselt, Arthur Guez, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. Learning values across many orders of magnitude. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4287–4295, 2016. URL http://papers.nips.cc/paper/6076-learning-values-across-many-orders-of-magnitude.pdf.

Richard Vinter. *Optimal Control*. Birkhauser, 2000.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782) :350–354, 2019.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3) : 279–292, May 1992.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4) :229–256, May 1992.

Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. Torcs, the open racing car simulator. *Software available at http ://torcs. sourceforge. net*, 4(6) :2, 2000.

Zhengyuan Yang, Yixuan Zhang, Jerry Yu, Junjie Cai, and Jiebo Luo. End-to-end multi-modal multi-task vehicle control for self-driving cars with visual perceptions. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 2289–2294. IEEE, 2018.

Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. *arXiv preprint arXiv :1910.01741*, 2019.

Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1851–1858, 2017.

Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2 : More deformable, better results. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9308–9316, 2019.

RÉSUMÉ

Dans cette thèse, nous abordons les défis de la conduite autonome en environnement urbain en utilisant des algorithmes d'apprentissage par renforcement profond de bout-en-bout, i.e. des données brutes des capteurs jusqu'au contrôle des actuateurs du véhicule.

L'apprentissage par renforcement (RL) est un des trois grands paradigmes de l'apprentissage automatique. Il se distingue de l'apprentissage supervisé par le fait que les agents apprennent par essai-erreur à partir d'un signal de récompense et non pas par simple supervision avec des paires entrée-label comme pour l'apprentissage supervisé, le type d'apprentissage le plus utilisé aujourd'hui dans les applications d'intelligence artificielle. Dans l'apprentissage par renforcement, on cherche explicitement à optimiser des séquences d'actions afin de maximiser le comportement à long terme. L'intérêt majeur du RL est que l'agent apprend de lui-même le comportement à suivre en explorant et en interagissant avec son environnement : on n'a donc pas besoin d'indiquer explicitement les actions à prendre.

Dans un premier temps, nous avons proposé un nouvel algorithme de renforcement fondé sur la fonction de valeur, Rainbow-IQN Ape-X, en combinant trois articles majeurs du domaine. Cet algorithme atteint des performances au niveau de l'état de l'art sur le benchmark Atari.

En utilisant cet algorithme de renforcement distribué, nous avons introduit les indices implicites, une nouvelle méthode permettant d'entraîner par renforcement des réseaux de neurones avec bien plus de paramètres et des entrées de plus grande dimension que les travaux précédents en DRL. Cette technique nous a ainsi permis de démontrer pour la première fois un algorithme de renforcement capable de conduire dans un simulateur complexe incluant des piétons, des véhicules et surtout des feux tricolores.

Finalement, nous avons utilisé toutes nos contributions précédentes pour effectuer de l'apprentissage par renforcement sur données réelles pour de la conduite en environnement urbain. L'idée fondamentale de notre approche est d'utiliser un simulateur fondé sur des images réelles pour réussir à entraîner des agents capables de généraliser aux données réelles.

MOTS CLÉS

apprentissage par renforcement, conduite autonome de bout en bout, apprentissage profond, vision par ordinateur

ABSTRACT

In this thesis, we address the challenges of autonomous driving in an urban environment using end-to-end deep reinforcement learning algorithms.

Reinforcement learning (RL) is one of the three paradigms of machine learning. It distinguishes itself from supervised learning by the fact that agents learn by trial and error from a reward signal and not by supervision with input-label pairs. In reinforcement learning, we explicitly seek to optimize sequences of actions in order to maximize long-term behavior. The major advantage of RL is that the agent learns the behavior to be followed by exploring and interacting with his environment : we therefore do not need to explicitly indicate the actions to be taken.

First, we proposed a new reinforcement algorithm, Rainbow-IQN Ape-X, by combining three major articles in the field of Value-based Reinforcement Learning. This algorithm achieves state-of-the-art performance on the Atari benchmark.

Using this distributed reinforcement algorithm, we introduced a new method coined implicit affordances, which allows to train by reinforcement neural networks with more parameters and larger inputs than previous works in DRL. This technique allowed us to demonstrate for the first time a reinforcement algorithm capable of driving in a complex simulator including pedestrians, vehicles and especially traffic lights.

Finally, we used all of our previous contributions to perform real data reinforcement learning for urban driving. The fundamental idea of our approach is to use a simulator based on real images to successfully train agents capable of generalizing to real data.

KEYWORDS

reinforcement learning, end-to-end driving, deep learning, computer vision