Identification de systèmes par réseaux de neurones pour la commande prédictive.

Antoine Mahé

NNT : 2020CSUP0010

CentraleSupélec

Ecole Doctorale IAEM

***« Informatique, Automatique, Electronique – Electrotechnique, Mathématiques»* – n°77**

Laboratoire du LORIA

THÈSE DE DOCTORAT

Spécialité de doctorat : Informatique (STIC)

Soutenue le 17 décembre 2020

par :

Antoine Mahé

Identification de systèmes par réseaux de neurones pour la commande prédictive

Composition du jury :

Directeur de thèse :	Matthieu GEIST	Pr, PU Université de Lorraine
Co-directeur de thèse :	Cédric PRADALIER	Pr, PU GeorgiaTech Lorraine
Président du jury :	Faïz BEN AMAR	Pr, PU, ISIR, Sorbonne Université
Rapporteurs :	Roland LENAIN	Dr, DR INRAE
	Emmanuel RACHELSON	Pr, PU ISAE-SUPAERO
Examinateurs :	Caroline CHANEL	Dr, IR ISAE-SUPAERO
Membres invités :	Hervé FREZZA-BUET	Pr, PU CentraleSupélec
	Alain DUTECH	Dr, CR Inria

UNIVERSITÉ DE LORRAINE

Résumé

École Doctorale IAEM, Département Informatique

doctorat de l'université de Lorraine

Identification de systèmes par réseaux de neurones pour la commande prédictive

par Antoine Mahé

Le développement de la robotique mobile permet la réalisation de tâches de plus en plus variées de façon automatisée. Cela représente un avantage majeur dans de nombreuses situations où l'intervention humaine n'est pas possible ou dangereuse. Cela est vrai pour la robotique aérienne qui est d'une grande utilité dans les tâches d'inspection. Le projet Grande Region rObotique aerienNE (GRoNe) dans lequel s'inscrit cette thèse, a pour objectif de développer les connaissances et les expérimentations sur ce sujet. Dans ce cadre, l'automatisation est un élément clé. Le développement d'algorithmes de contrôle efficaces est une étape importante en ce sens. La réalisation de ces algorithmes requiert la conception de modèles précis des systèmes afin de prédire leur évolution. Le but d'un algorithme de contrôle est d'établir une stratégie permettant d'obtenir les commandes à envoyer au système en fonction de son état dans le but d'atteindre un objectif. Dans le cadre de la robotique mobile, les contrôleurs à commandes prédictives ont démontré de nombreux avantages. Il s'agit d'une méthode utilisant un modèle du système afin de prédire le résultat de l'application de commandes sur ce dernier. Ainsi, il est possible de choisir les meilleures commandes pour une trajectoire voulue. La version de cet algorithme utilisant l'intégrale de parcours pour le choix de la commande est particulièrement pertinente dans le cas de fonction de coût complexe. L'algorithme est basé sur l'utilisation d'un modèle de la dynamique du système et son efficacité est directement liée à la précision de la modélisation. Le problème de la modélisation en robotique est traité par l'identification de système. Il s'agit de déduire le modèle de l'observation des données du système en fonctionnement. L'apprentissage automatique est utilisé de plus en plus fréquemment dans ce contexte, notamment du fait de la facilité à modéliser des systèmes avec des non-linéraités complexes par des réseaux de neurones. Afin de modéliser les systèmes robotiques considérés, en simulation mais aussi sur le terrain, des corpus de données des robots en fonctionement ont été collectés. Différentes architectures de réseaux de neurones ont été comparées pour la conception de modèles à partir de ces corpus. Cependant les échantillons collectés en démonstration ne correspondent pas nécessairement à la situation dans laquelle les modèles doivent être utilisés. En effet, les exemples sont collectés dans certaines conditions et le but du contrôleur est d'utiliser ces informations en vue de réaliser de nouvelles tâches. Dans un corpus de données, certains exemples sont plus utiles au processus d'apprentissage que d'autres. Les exemples nombreux de situations courantes deviennent rapidement redondants alors que les échantillons de situations plus complexes sont rares. Une solution à ce problème est de mettre en avant certains exemples durant le processus d'apprentissage. En priorisant certains éléments, l'entraînement est rendu plus efficace. La sélection des échantillons à prioriser est la partie délicate. Pour cela, deux méthodes sont comparées : l'une utilise comme mesure de l'importance d'un exemple l'erreur du modèle sur ce cas, l'autre utilise une approximation

de l'effet de cet exemple sur l'apprentissage. Ces méthodes permettent de réaliser de nouveaux modèles des systèmes robotiques. La réalisation de modèles n'est pas une fin en soit, il s'agit d'une composante importante de l'algorithme de contrôle. Il importe donc d'étudier les différents modèles au sein du contrôleur global. L'application de ce contrôleur à des systèmes robotiques tels qu'un drone et un bateau, tant en simulation qu'en cas réel, permet d'étudier les avantages de son utilisation. Finalement un modèle priorisé est utilisé au sein d'un algorithme à commande prédictive sur le bateau réel afin de réaliser avec succès un suivi de berge sur un lac artificiel.

Remerciements

Je souhaite tout d'abord remercier mon directeur et mon co-directeur de thèse, Matthieu Geist et Cédric Pradalier, pour l'opportunité qu'ils m'ont offerte, pour leurs nombreux conseils, pour leur patience et leur disponibilité.

Je tiens à remercier l'initiative FEDER INTERREG VA qui a financé ma thèse dans le cadre du projet GRoNe ainsi que la région Grand Est.

Je souhaite aussi remercier les laboratoires au sein desquels j'ai travaillé : l'antenne de Metz du LORIA à CentraleSupélec ainsi que l'équipe DREAM à Georgia Tech Lorraine.

J'aimerais remercier l'ensemble des collègues avec qui j'ai eu le plaisir de travailler, de discuter et d'échanger. En particulier, Antoine Richard, Assia Benbihi, George Chahine, Othmane Ouabi et Stéphanie Aravecchia de l'équipe DREAM pour les nombreux échanges et collaborations.

Je souhaite aussi adresser un grand merci à ma famille pour son soutien, en particulier à ma mère pour son aide dans les moments difficiles.

J'ai une pensée pour tous les amis avec qui j'ai partagé des moments pendant ces trois ans et demi. Plus particulièrement, merci à Thomas Fitoussi, pour avoir ouvert la voie, pour être toujours un si bon ami après si longtemps et pour continuer de débattre de façon aussi enrichissante.

Enfin je voudrais remercier le cercle de lecture du campus de Metz de CentraleSupélec, en particulier Yann Liber pour organiser ces échanges riches.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Ce chapitre introduit les motivations du travail présenté dans cette thèse. Puis une description des principales contributions de ce travail est présentée. Enfin une description de l'organisation du mémoire est détaillée.

1.1 Contexte et motivation

L'utilisation de véhicules autonomes est de plus en plus répandue pour l'accomplissement de tâches variées. Par exemple, l'agriculture de précision, l'inspection de bâtiments ou la surveillance d'environnements naturels bénéficient grandement de l'utilisation de véhicules autonomes aériens ou nautiques. Cependant, dans l'extrême majorité des cas, les systèmes sont commandés par des opérateurs humains, ce qui limite leur potentiel. Par exemple, l'inspection de mines souterraines, pour lesquelles les conditions de télécommunications sont complexes, pose un risque pour la sécurité des opérateurs. Il existe donc de fortes motivations pour améliorer l'autonomie des systèmes robotiques mobiles utilisés pour ce type de tâches, en particulier les véhicules aériens autonomes, en anglais *Unmanned Aerial Vehicles* (UAV), et les véhicules autonomes de surface, en anglais *Unmanned Surface Vehicles* (USV).

Le projet GRoNe [1], dans lequel s'inscrit cette thèse, vise au développement de la robotique aérienne dans la Grande Région [2]. L'objectif du projet est la création d'un réseau transfrontalier basé sur le développement de la connaissance et de l'expérimentation sur le sujet, en s'appuyant sur les ressources technologiques et scientifiques de ce domaine présentes dans la Grande Région. Le travail présenté ici contribue aux aspects de développement de la connaissance et de l'expériementation en matière de robotique aérienne. Les concepts développés dans ce cadre ne se limitent cependant pas à la robotique aérienne et sont aussi montrés dans le cadre de bateaux autonomes.

L'enjeu du contrôle des robots est de concilier la complétion de missions avec des objectifs précis tout en respectant les contraintes imposées par l'environnement. Une partie des difficultés vient de la complexité de ces robots. La variété et le nombre d'entrées-sorties sont des points problématiques, leur dynamique complexe en est un autre. Une méthode de contrôle couramment utilisée est la commande prédictive, en anglais *Model Predictive Control* (MPC), qui s'appuie sur un modèle dynamique du système afin d'optimiser une fonction de coût sur un horizon déterminé. Ce type de méthodes a rencontré un grand succès ces dernières années. En effet un nombre important d'industries ont indépendamment adopté cette famille de contrôleurs en

1. https://interreg-grone.eu/fr/accueil/
2. La Grande Région est un groupement européen de coopération territoriale regroupant des divisions territoriales allemandes, belges, et françaises et centré sur le Luxembourg.

particulier du fait de son succès à gérer les problèmes multivariables en comparaison des autres méthodes. Ce succès s'explique par la capacité de ce type de méthodes à fonctionner pour des systèmes complexes avec des contraintes fortes et une multitude d'entrées-sorties. De plus la facilité d'incorporer à la fois les objectifs et les contraintes d'une mission dans la fonction de coût permet une adaptation facile de ce contrôle à différents contextes. Les méthodes de commandes prédictives nécessitent de simuler le fonctionnement du système, ce qui requiert la conception d'un modèle du robot.

1.1.1 Modélisation en robotique

Il existe de nombreuses méthodes pour modéliser un système. Les deux grandes catégories pour ces méthodes sont : approche en boîte blanche et approche en boîte noire. L'approche dite boîte blanche (ou transparente) fait référence à l'utilisation de connaissances *a priori* du système. Il s'agit d'utiliser une modélisation de la physique du robot pour obtenir un modèle mathématique du système. Le grand avantage de ces méthodes est la compréhension du modèle obtenu et ainsi la capacité à expliquer le comportement de ce dernier. Cependant, la conception de ce modèle basée sur la compréhension physique du système n'est possible que grâce à la simplification et à l'approximation du système réel. Par conséquent, il est souvent difficile d'obtenir des modèles précis pour les robots à la dynamique complexe. D'un autre coté, les approches en boîte noire s'affranchissent de connaissances *a priori* du système et s'appuient sur l'observation des données de ce dernier en fonctionnement afin de déduire l'évolution de son état. Ces approches permettent d'obtenir des résultats plus précis que les approches précédentes. Cependant, elles ne présentent pas les mêmes avantages d'explicabilité que permettent les approches utilisant des connaissances de la physique du système. De plus elles requièrent la collecte de corpus de données du système en fonctionnement, ce qui peut être problèmatique.

1.1.2 Description des robots utilisés

Le cas de robot mobile évoluant dans des fluides (drone, bateau) est un cas pour lequel la dynamique des robots est complexe : mouvement dans un fluide, non-linéarité des actionneurs, retard dans la dynamique. De plus, la précision des modèles pour l'exécution des tâches d'inspection est de grande importance. Par conséquent, les approches concentrées sur l'étude de l'observation des données du système en fonctionnement sont préférables. Il s'agit du champ d'études de l'identification de systèmes en boîte noire. En particulier les deux systèmes robotiques suivants, le drone bebop 2 de Parrot [3], illustré dans la figure 4.7 et le Héron de Clearpath Robotic [4] , illustré en figure 4.10, sont étudiés.

Le premier système d'étude considéré est le drone bebop 2 de Parrot. Il s'agit d'un drone du commerce dont la commande bas niveau des moteurs n'est pas accessible. En conséquence, lors des études de modélisation, le système identifié comprend à la fois le drone en lui-même et le contrôleur bas niveau. Afin d'acquérir des données de bonne précision, le drone est équipé d'un *Global Positionning System* (GPS) supplémentaire, le M+ de EMLID. En effet ce dernier permet de corriger le problème de dérive du positionnement GPS grâce à sa coordination avec un GPS de base fixe, il s'agit d'un

3. https://www.parrot.com/fr/
4. https://clearpathrobotics.com/heron-unmanned-surface-vessel/

FIGURE 1.1 – Le drone Parrot utilisé, équipé du GPS RTK.

système GPS *Real Time Kinematic* (RTK). L'ajout de matériel sur le drone n'est pas anodin. En effet, le drone pèse 1.477 kg et n'est pas prévu pour soulever des charges. Il n'a donc pas été possible d'embarquer la batterie permettant d'alimenter le GPS sur le drone et une modification de ce dernier pour alimenter le GPS à l'aide de la batterie du drone a été nécessaire. L'étude de ce drone est intéressante pour évaluer l'applicabilité des méthodes sur une plateforme aérienne grand public. Cependant la présence du contrôleur de bas niveau simplifie le problème du contrôle du drone du fait de son effet de linéarisation sur la dynamique du système. Cela limite donc la portée des solutions démontrées sur cette plateforme.

En revanche, le second système considéré, le bateau Heron de Clearpath Robotic, présente une dynamique beaucoup plus complexe. D'une part, l'inertie du bateau lors de son déplacement dans l'eau implique un retard dans la dynamique. D'autre part, les non-linéarités importantes des moteurs des turbines du bateau ont une grande influence sur la dynamique. Ce véhicule autonome de surface est un catamaran équipé de deux turbines utilisées à la fois pour la propulsion et la direction. De même que le drone, le bateau est équipé d'un GPS RTK. De plus, tous les calculs nécessaires au contrôle du bateau sont faits à l'aide de l'informatique embarquée. L'informatique présente à bord est séparée en deux composants principaux. D'une part une carte équipée d'un *atom* Intel est utilisée pour les calculs bas niveau. D'autre part une carte Xavier de Nvidia est utilisée pour les opérations coûteuses en calcul.

1.1.3 Identification

L'objectif de l'identification de système est de construire un modèle qui permet de prédire son évolution. Soit l'état du système $X(t)$ regroupant l'ensemble des variables permettant de décrire le système et soit la commande $U(t)$ comprenant l'ensemble des variables de commande du système. L'objet de l'étude est l'évolution du système au cours du temps, plus précisément la prédiction de l'état du système futur connaissant ses états passés et les commandes qui lui sont appliquées. Soit F le modèle recherché, l'équation (1.1) le décrit formellement pour un historique de n pas.

$$\begin{cases} \{X(t), X(t-dt), ...X(t-n*dt)\} \\ \{U(t), U(t-dt), ...U(t-n*dt)\} \end{cases} \xrightarrow{F} X(t+dt). \tag{1.1}$$

FIGURE 1.2 – Bateau Héron de clearpath utilisé dans les expériences

Le principe de l'identification est de déduire le modèle recherché de l'observation du système en fonctionnement. Il s'agit donc dans un premier temps de collecter les données du robot en action puis d'utiliser une méthode de régression pour calculer le modèle. Ce type de modèle pourra ensuite être utilisé dans des contrôleurs comme la commande prédictive décrite au chapitre 3.

L'approche classique de l'identification des systèmes en boîte noire est la famille des méthodes autorégressives de type *Auto Regressive Moving Average* (ARMA) et ses extentions (ARX, ARMAX) (LJUNG, 1998). Ces méthodes sont utilisées depuis des décennies avec succès. Considérant la commande $U(t)$ et la sortie $X(t)$, le modèle ARMA du système est donné par l'équation aux différences finies suivante :

$$X(t) + \sum_{k=1}^{p} a_k X(t-k) = \sum_{k=1}^{q} b_k U(t-k). \tag{1.2}$$

Généralement, l'objectif est de déterminer la valeur de la sortie du système au prochain pas de temps étant donné la sortie précédente, l'entrée et les paramètres $\{a_1, ..., a_p, b_1, ...b_q\}$:

$$X(t) = -\sum_{k=1}^{p} a_k X(t-k) + \sum_{k=1}^{q} b_k U(t-k). \tag{1.3}$$

La linéarité du modèle permet de facilement calculer les paramètres optimaux en utilisant une régression des moindres carrés sur une série d'observations faites du système en fonctionnement.

Malheureusement, tous les systèmes ne sont pas linéaires. Les frictions et les chocs sont des exemples courants de non-linéarité. Or, l'extension des modèles régressifs à des systèmes non-linéaires est complexe. Il est nécessaire de faire appel à des méthodes

différentes, méthodes qui doivent se spécialiser sur les systèmes étudiés. Pour régler les paramètres de ces méthodes spécialisées, il faut faire appel à une expertise des systèmes en question.

Il serait souhaitable d'avoir une méthodologie applicable aux différents types de robots de façon équivalente. C'est l'intérêt d'utiliser des méthodes d'apprentissage automatique. Ces méthodes de modélisation ne s'adaptent pas toujours facilement au contexte de la robotique mobile. En particulier l'extention des modèles à de nouvelles situations est un problème délicat.

1.2 Contributions

Le problème de la modélisation d'un système complexe est central dans la mise en place d'un contrôleur efficace. Le problème de la modélisation de robot s'appuyant sur l'analyse des données du système en fonctionnement a été largement étudié. De nombreuses difficultés sont rencontrées dans l'application de ces méthodes dans le cadre du contrôle de robots mobiles. En particulier la qualité et le déséquilibre des corpus de données sont fréquemment des obstacles importants. L'objectif de cette thèse est d'étudier la pertinence de l'utilisation de l'apprentissage automatique couplée avec des méthodes d'échantillonnages préférentiels pour le contrôle de systèmes complexes. Ainsi les contributions de cette thèse sont les suivantes :

— une étude aussi exhaustive que possible de l'identification de systèmes par réseaux de neurones ;
— l'utilisation de méthodes de priorisation dans le cadre de l'identification de systèmes basées sur l'apprentissage automatique afin de traiter le problème du déséquilibre des corpus ;
— l'étude de différentes méthodes d'échantillonage préférentiel est faite pour plusieurs corpus d'identification en simulation mais également sur des systèmes réels ;
— Les modèles sont évalués dans un contrôleur à commande prédictive, appliqué sur différentes tâches.

Les travaux effectués durant la thèse ont donné lieu à un certain nombre de contributions :

— A. Mahé, C. Pradalier et M. Geist (2018). « Trajectory-control using deep System Identication and Model Predictive Control for Drone Control under Uncertain Load. » In : *2018 22nd International Conference on System Theory, Control and Computing (ICSTCC)*, p. 753-758
— Antoine Mahé et al. (2019). « Importance Sampling for Deep System Identification ». In : *2019 19th International Conference on Advanced Robotics (ICAR)*. IEEE, p. 43-48

Ainsi qu'un certain nombre de travaux soumis et en cours d'évaluation :

— Antoine Mahé et al. (2020). « Evaluation of prioritized deep system identification on a path following task »
— Antoine Richard et al. (2020). « Fast Paced Shore Following Using Data Driven Model Predictive Control. »

1.3 Organisation du mémoire

Le chapitre 2 présente l'état de l'art. Dans un premier temps, la robotique mobile y est introduite, en particulier dans le cadre des systèmes aériens et nautiques qui sont

utilisés dans la suite de l'étude. Ensuite les travaux utilisant le contrôle prédictif qui ont inspiré ce travail sont présentés. Une présentation des techniques d'apprentissage automatique pertinentes dans le cadre de l'identification de système est faite ensuite. Enfin une présentation de la priorisation dans le cadre de l'apprentissage est faite.

Le chapitre 3 s'intéresse à la commande de robots mobiles. Le contrôle à commande prédictive présentant de nombreux avantages dans ce domaine y est détaillé. En particulier une version de cette méthode peu contraignante pour la fonction coût est étudiée. Ce qui nous permet de l'utiliser avec des fonctions de coût complexes encodant à la fois les contraintes et les objectifs des tâches considérées. La méthode en question est la commande prédictive à intégrale de parcours. Les différents éléments de cet algorithme sont discutés. Le contrôleur est implémenté pour plusieurs systèmes. Le test est d'abord fait sur un drone en simulation, puis la modélisation est étendue à un corpus de données collectées sur le terrain. Ensuite un système aux non-linéarités plus marquées est considéré : un catamaran autonome. L'influence des différents paramètres du contrôleur est évaluée empiriquement sur une tâche simulée de suivi de piste.

Le chapitre 4 étudie la modélisation de systèmes complexes qui est nécessaire au contrôle à commande prédictive. En particulier l'utilisation de l'apprentissage automatique dans le cadre de l'identification de systèmes est considéré. La pertinence des différentes architectures de réseaux couramment utilisées est étudiée et discutée. De plus, cette méthode est appliquée à l'identification pour plusieurs systèmes (un drone et un bateau) en simulation et aussi en cas réels. Enfin, l'application de ces modèles dans le cadre du contrôle des systèmes est considérée sur des tâches de suivi de piste.

Le chapitre 5 décrit des mécanismes d'échantillonage préférentiel utilisés pour traiter les problèmes liés au déséquilibre des corpus. Ces méthodes sont ensuite mises en œuvre pour l'identification des systèmes. Les résultats obtenus par cette méthode sont comparés aux résultats précédents à la fois en terme de précision des prédictions à un et plusieurs pas et aussi en terme de performance des contrôleurs utilisant ces modèles. Les différents hyperparamètres des méthodes de priorisation sont discutés.

Le chapitre 6 résume les avancées proposées et met en avant les pistes de recherche qui semblent pertinentes de poursuivre.

Chapitre 2

État de l'art

Les progrès de l'automatisation ont permis le développement d'une industrie de plus en plus efficace. Alors que l'humanité cherche sans cesse à améliorer sa condition en exploitant au mieux les ressources à sa disposition, les progrès de la robotique ouvrent la possibilité d'exploiter des ressources jusque-là inacessibles, telles que celles présentes dans les fonds sous-marins ou dans l'espace. De même, la possibilité de déléguer de plus en plus de tâches à des systèmes robotiques présente un fort potentiel. Pour réaliser ce potentiel, des algorithmes de contrôle sont continuellement développés et améliorés. Alors que les puissances de calcul embarquables à bord des systèmes autonomes sont en constante évolution (Amodei et Danny Hernandez, 2019), les théories prometteuses du 20e siècle (Turing, 1950) prennent leur essor (Silver et al., 2016 ; Wang et al., 2016). Dans un premier temps les avancées des recherches en matière de robotique mobile sont présentées, en particulier dans les domaines aérien et nautique pour lesquels des expériences sont réalisées. Ensuite, le problème de la modélisation des systèmes est abordé sous l'angle de l'identification. Puis les récents progrès de la commande prédictive qui s'appuie sur la modélisation précise de la dynamique des systèmes sont explorés. Ensuite les progrès des applications de l'apprentissage automatique, notamment en relation avec la robotique mobile sont examinés. Finalement, la littérature concernant les problématiques d'échantillonnage préférentiel appliqué à l'apprentissage automatique est évoquée.

2.1 Robotique mobile

Les systèmes considérés ici sont des plateformes mobiles, c'est-à-dire capables de se mouvoir d'elles-mêmes dans l'espace. L'objectif est d'être capable de contrôler ce déplacement, c'est-à-dire de maîtriser l'ensemble des positions futures du système considéré. Le sujet pourrait être abordé dans le cas général. En effet les mêmes techniques de contrôle peuvent être appliquées à d'autres systèmes dynamiques (contrôle de réaction chimique, de procédé d'usinage, etc.). Cependant le propos se restreint ici aux plateformes de robotique mobile. L'évolution de leur position au cours du temps, appelée trajectoire, est au cœur de ce travail. L'objectif est d'être capable de trouver les commandes à envoyer à la plateforme robotique pour assurer le suivi d'une trajectoire donnée.

2.1.1 Robotique aérienne

Les robots aériens, appelés aussi drones ou UAV, sont un objet d'étude intéressant avec beaucoup de potentiel. En effet, de telles plateformes rendent possible l'automatisation de nombreuses tâches jusque là effectuées par des opérateurs humains, notamment dans le cadre de l'inspection. Cette automatisation est particulièrement pertinente lorsque l'intervention humaine est difficile ou dangereuse. L'agriculture de

précision (Alsalam et al., 2017 ; Pretto et al., 2019) et la surveillance d'infrastructures (Mader et al., 2016) (canalisations pétrolières, routes, bâtiment, mines désafectées, etc.) sont des exemples d'application de la robotique aérienne.

Il existe une grande variété de plateformes aériennes : ballon, dirigeable, planneur, avion, hélicoptère et quadrirotor. Chaque plateforme a ses propres caractéristiques, inconvénients et avantages. Le périmètre de ce travail se restreint aux quadrirotors. En effet, ce type de robot présente des avantages intéressants. La capacité de décoller et d'attérir verticalement en fait l'un des systèmes aériens les plus faciles d'utilisation et le développement récent de la commercialisation de ces drones les a rendus très accessibles. De plus, leur capacité à faire du surplace leur permet de remplir des tâches d'inspection autrement difficiles (Gupte, Mohandas et Conrad, 2012). L'enjeu majeur de l'utilisation de drones est leur capacité à accomplir les missions en autonomie. L'amélioration des algorithmes de contrôle est l'objet du travail présenté ici.

2.1.2 Robotique nautique

Les robots se déplaçant sur les surfaces aquatiques sont aussi très utilisés pour des tâches d'inspection et de surveillance (Dunbabin, Grinham et Udy, 2009 ; Groves et al., 2019). Cependant ces plateformes nautiques présentent des problématiques de modélisation différentes des systèmes aériens (Sonnenburg et Woolsey, 2013). Les caractéristiques des USV sont très différentes des quadrirotors. En particulier, les problèmatiques d'inertie sont plus prépondérantes dans l'eau que dans l'air. Cela implique des différences importantes dans les modèles nécessaires au contrôle de ces systèmes. L'application des méthodes d'identification de système et de contrôle à des robots ayant des caractéristiques aussi différentes permet d'évaluer ces méthodes de façon plus complète.

2.1.3 Outils logiciels

Les systèmes robotiques présentent des conditions particulières en termes de logiciels : ils fonctionnent avec des contraintes liées au monde physique, des contraintes de temps réel importantes et doivent gérer une diversité de capteurs et d'effecteurs avec des caractéristiques physiques spécifiques. Pour faciliter la recherche dans ce domaine, plusieurs environements de développement spécifiques ont été développés (Kramer et Scheutz, 2007). En particulier l'utilisation de *middelware*, couche logicielle intermédiaire entre les algorithmes de haut niveau et le robot physique, est d'une grande utilité (Elkady et Sobh, 2012). Il s'agit de faire le lien entre les commandes abstraites (par exemple : "accélérer") et leur réalisation physique (par exemple : envoi de courant dans les moteurs). Dans cette étude, une utilisation importante de la structure *Robotic Operating System* (ROS) (Quigley et al., 2009) a été faite. L'objectif de cette structure logicielle est de proposer une organisation équivalente aux systèmes d'opérations des ordinateurs classiques, adaptée aux problèmatiques rencontrées en robotique. Elle ne remplace pas les systèmes d'opérations classiques mais fonctionne en parallèle en fournissant des outils supplémentaires spécifiques au contexte robotique. Ainsi, l'utilisation de ROS fournit de nombreux services lors du développement de logiciels, notamment : l'abstraction matérielle, le contrôle bas niveau de périphérique, l'échange de messages entre processus et la gestion de paquets. L'utilisation de cette structure permet de rapidement intégrer différents systèmes et capteurs en

s'appuyant sur les bibliothèques logicielles et les dépôts de paquets partagés et mis à jour par les contributeurs.

Un autre aspect logiciel très important lors du développement d'application robotique est la simulation. En effet, la réalisation de ces systèmes complexes nécessite un nombre important de tests. Les contraintes de sécurité des personnes et du matériel ainsi que la complexité du monde réel rendent les expériences sur le terrain coûteuses. Il est donc important de pouvoir tester les algorithmes le plus précisement possible. La possibilité de prototyper des algorithmes rapidement est étroitement liée à la capacité de les tester efficacement. De nombreux logiciels de simulation sont disponibles (Staranowicz et Mariottini, 2011). En particulier V-REP (Rohmer, Singh et Freese, 2013) et Gazebo (Koenig et Howard, 2004) sont régulièrement utilisés avec ROS. Le travail présenté ici a été réalisé en utilisant Gazebo pour lequel des paquets permettant de modéliser précisement les robots étudiés étaient disponibles.

2.2 Contrôle prédictif

Pour permettre plus d'autonomie, les algorithmes de contrôle sont continuellement développés et améliorés. En particulier, la commande prédictive a été utilisée pour le contrôle des drones dans de nombreuses situations avec succès (Naegeli et al., 2017 ; Dentler et al., 2016). La commande prédictive est basée sur l'optimisation en temps réel d'une fonction de coût, plus précisement il s'agit d'optimiser le coût total de la trajectoire pour un horizon donné. Une version de cette méthode dite à "intégrale de parcours" décrite par Williams et al., 2016 permet de prendre en compte des fonctions de coût de plus grande complexité. La flexibilité dans la conception de la fonction de coût permet de programmer à la fois les objectifs et les contraintes. Une explication détaillée de la construction de la fonction de coût est donnée dans le chapitre 3. Cela est particulièrement utile dans le contexte de mission de véhicules volants pour lesquels les contraintes de sécurité entrent souvent en compétition avec les objectifs de la mission.

Habituellement, lorsqu'il s'agit de suivi de lignes, les contrôleurs réactifs (Pêtrès, Romero-Ramirez et Plumet, 2011) sont connus pour bien fonctionner. Cependant, il suppose que l'inertie des systèmes soit inexistante ou que la dynamique soit connue. Malheureusement cette hypothèse n'est pas toujours vérifiée pour les robots autonomes. Par exemple, le véhicule autonome de surface Heron de Clearpath robotic présenté dans l'introduction peut nécessiter jusqu'à 20 mètres pour s'arrêter entièrement de lui-même en partant d'une vélocité de 1 m/s. Par conséquent, la dynamique du système doit être modélisée. De nombreux contrôleurs à commande prédictive (Velasquez et al., 2020 ; Lenain et al., 2006 ; Lucet, Lenain et Grand, 2015) pourraient être appliqués. Cependant ils s'appuient sur des modèles dynamiques connus et différenciables. De plus, même si ces contrôleurs fonctionnent bien sur des systèmes linéaires (Qin et Badgwell, 2003), ils ont des difficultés lors de leur application à des systèmes non-linéaires (Mayne, 2014). En général, ces méthodes s'appuient donc sur des versions linéarisées des systèmes et ne sont applicables que sur des sous-parties de l'espace d'action. Une autre solution est d'utiliser la planification en maillage (Griffith, Chahine et Pradalier, 2017 ; Pivtoraiko, Knepper et Kelly, 2009) : étant donné un ensemble préenregistré de trajectoires, la plus pertinente est choisie pour les étapes suivantes. Cette approche présente trois inconvénients

majeurs : elle ne fonctionne pas de façon fiable à hautes vitesses ; elle requiert beaucoup de réglages ; elle peut échouer laissant le robot complètement bloqué. C'est pour répondre à ces problèmes que l'utilisation de la commande prédictive à intégrale de parcours (Williams et al., 2017a ; Williams et al., 2016) est choisie dans ces travaux. Ce contrôleur s'appuie sur une recherche Monte Carlo (Browne et al., 2012) pour trouver la séquence de commandes à envoyer au système en simulant le comportement futur du robot.

2.3 Modélisation et apprentissage

Le problème de la modélisation des systèmes est un élément crucial du contrôle robotique (Khalil et Dombre, 2004). Il est soit abordé sous l'angle des équations physiques du système (approche de la catégorie boîte blanche), soit sous l'angle de l'analyse des données issues de l'observation de l'évolution du système en fonctionnement (approche en boîte noire). Depuis les années 1970 ARMA (Ljung, 1998) est la technique de choix pour la plupart des problèmes d'identification de système en boîte noire. De tels modèles ont l'avantage d'être particulièrement simples à comprendre et à implémenter. Ils s'appuient sur la résolution d'équations différentielles linéaires discrètes. La linéarité de ces équations permet le calcul des paramètres du système via la méthode des moindres carrés. Par conséquent la recherche d'un modèle ARMA n'est pas coûteuse en calcul. Cela permet de faire des recherches systématiques sur les métaparamètres tels que les ordres du modèle.

L'apprentissage automatique a été utilisé avec beaucoup de succès ces dernières années pour de nombreux problèmes relevant du contrôle (Hwangbo et al., 2017 ; Zhang et al., 2015). Les réseaux de neurones ont montré des capacités à modéliser des systèmes complexes, ce qui en fait des outils de choix pour l'implémentation de contrôleurs prédictifs. Cependant, pour être capable d'entraîner ces réseaux, il est nécessaire d'utiliser des quantités importantes de données. Cela implique un grand nombre de démonstrations du système dans son environnement.

2.3.1 Apprentissage automatique

Les essais récents d'identification de système en boîte noire utilisant de l'apprentissage profond ont montré des résultats impressionnants, à la fois pour des systèmes linéaires et des systèmes non-linéaires (Zhang et al., 2016 ; Gonzalez et Yu, 2018). Ces travaux s'appuient sur des architectures telles que les perceptrons multicouches, en anglais *Multi Layer Perceptron* (MLP), et les réseaux récurrents à mémoire court et long termes, en anglais *Long Short-Term Memory* (LSTM) (Hochreiter et Schmidhuber, 1997).

2.3.2 Apprentissage automatique appliqué à l'identification

L'entraînement de réseaux de neurones a été utilisé pour l'identification de systèmes durant ces trente dernières années (Narendra et Parthasarathy, 1992 ; Narendra et Mukhopadhyay, 1992). Les avancées récentes à la fois en termes de matériel et d'architecture (Amodei et Danny Hernandez, 2019) ont ramené sur le devant de la scène ces méthodes d'apprentissage.

L'utilisation de l'apprentissage automatique pour la modélisation de systèmes robotiques est souvent mise en avant lorsque les méthodes d'identification de types ARMA ne sont plus suffisantes. C'est en particulier vrai lorsque les systèmes sont complexes et présentent des non-linéarités marquées. Schaal, Atkeson et Vijayakumar, 2002 utilisent l'entraînement de réseaux de neurones pour construire le modèle dynamique inverse du robot humanoïde DB. Dans Angelova et al., 2007 l'apprentissage automatique est utilisé pour prédire le glissement d'un robot mobile. Wehbe, Hildebrandt et Kirchner, 2017 utilisent l'apprentissage automatique pour modéliser la dynamique d'un robot sous-marin.

Cependant, peu de travaux ont proposé de comparer la pertinence des différentes structures disponibles pour des tâches d'identification. Lorsque ces comparaisons sont faites (Ogunmolu et al., 2016) elles s'appuient sur de vieux corpus tels que DaISy (De Moor et al., 1997) qui sont très petits et contiennent trop peu d'exemples pour être utilisés pour entraîner correctement les architectures complexes de réseaux de neurones. Les différentes architectures de réseaux de neurones communément appliquées à l'identification de systèmes sont présentées ci-après.

Architectures

L'identification de système et la prévision de séries temporelles sont étroitement liées d'un point de vue de la structure des données. Dans les deux cas, une séquence d'entrées est utilisée pour prédire une donnée future (ou une séquence de données futures). Dans le domaine de l'identification de modèles, des architectures de type *Multi Layer Perceptron* (MLP) ou plus récemment *Long Short-Term Memory* (LSTM) (Hochreiter et Schmidhuber, 1997) sont fréquentes. Cependant, il y a de nombreuses approches dans le domaine de la prévision temporelle qui pourraient aussi s'appliquer à l'identification. Ces approches comprennent notamment les réseaux convolutionnels à une dimention (LeCun, Bengio et al., 1995), leur variante à plusieurs têtes *Multi-head Convolutional Neural Network* (MH-CNN), et la large famille des réseaux récurrents qui inclut les *Gated Recurrent Unit* (GRU) (Cho et al., 2014) et les LSTM et leurs variations.

Le MLP est l'une des architectures de neurones les plus utilisées dans l'identification de systèmes. Il s'appuie sur l'empilement de couches denses aussi appelées couches entièrement connectées. Ces couches se composent de multiplications matricielles et d'additions de biais. Elles sont souvent complétées par une fonction d'activation, de normalisation et/ou de régularisation.

Les réseaux convolutionnels à une dimension sont moins connus que leur équivalent en deux dimensions mais ils ont un avantage sur les simples MLP. Par construction les réseaux convolutionnels à une dimension traitent leurs séquences d'entrées de façon ordonnée. Ce traitement temporel est intéressant car son comportement ressemble à celui des réseaux récurrents sans présenter les problématiques d'entraînement et de réglages de ces derniers. Leurs principaux inconvénients sont le nombre d'hyperparamètres qui rend difficile la conception, le temps conséquent d'entraînement et la lenteur de leurs temps d'inférence comparés au MLP.

Les réseaux convolutionnels multi-têtes sont une extension des réseaux convolutionnels à une dimension. Le principal avantage est leur flexibilité. Dans la plupart des

cas, ces réseaux ont une tête par variable d'entrée. Ces têtes sont composées de réseaux convolutionnels à une dimension ; ces réseaux sont spécialement réglés pour les variables que les têtes traitent. Les différents paramètres de ces sous-réseaux convolutionnels à une dimension sont adaptés à chaque entrée individuellement. Les résultats produits par ces têtes sont ensuite concaténés ensemble et traités par une couche dense. Le principal défaut de cette architecture est son principal avantage : le très grand niveau de personnalisation requis pour chaque tête réglée individuellement. En effet la flexibilité obtenue grâce à cette architecture se paie par la difficulté de sa mise au point.

Récemment, les architectures récurrentes ont permis de définir un nouvel état de l'art dans le Traitement Automatique des Langues. Ces architetures ont aussi été utilisées dans le cadre de la prévision des séries temporelles ainsi que pour résoudre le problème de données manquantes (Che et al., 2018). Les réseaux récurrents sont une classe de réseaux de neurones qui dépendent explicitement du temps. Ils utilisent leur mémoire interne pour évaluer le contexte. C'est cette contextualisation qui leur a permis de traiter les phrases d'une langue efficacement. En effet, lors de la lecture, l'interprétation d'un mot est conditionné par les mots qui l'entourent. Cet avantage qui les rend pertinent pour le Traitement Automatique des Langues est aussi valable dans le cadre de l'identification du modèle où la dynamique se déduit de l'observation des vitesses autour d'un instant. La figure 2.1 illustre deux représentations de ces réseaux récurrents : à gauche la version enroulée, à droite la version déroulée. L'illustration montre comment le réseau A produit la sortie h à partir de l'entrée x et de son traitement de l'entrée précédente. Le principal inconvénient de ces architectures récurrentes est la difficulté de la gestion de la mémoire cachée qui est au cœur de leur capacité à évaluer le contexte.

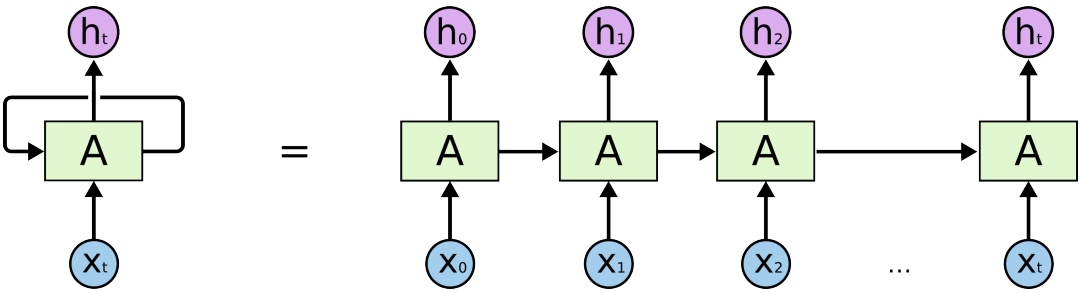


Figure 2.1 – Structure d'un réseau de neurones récurrent [1].

Les réseaux à mémoire court et long termes ont été développés afin d'apprendre les dépendances à long terme. Ces réseaux ont la particularité d'utiliser un procédé de mise-à-jour de leur état caché laissant passer plus ou moins d'informations. La figure 2.2 propose une illustration de ces réseaux en reprenant les notations de la figure précédente.

Les réseaux de neurones GRU, appelés en français réseaux récurrents à portes, introduits par Cho et al., 2014, sont similaires aux réseaux à mémoire court et long termes précédemment présentés. Ils proposent une simplification du procédé de mise

1. Image extraite de http://colah.github.io/posts/2015-08-Understanding-LSTMs/?3source=post_page

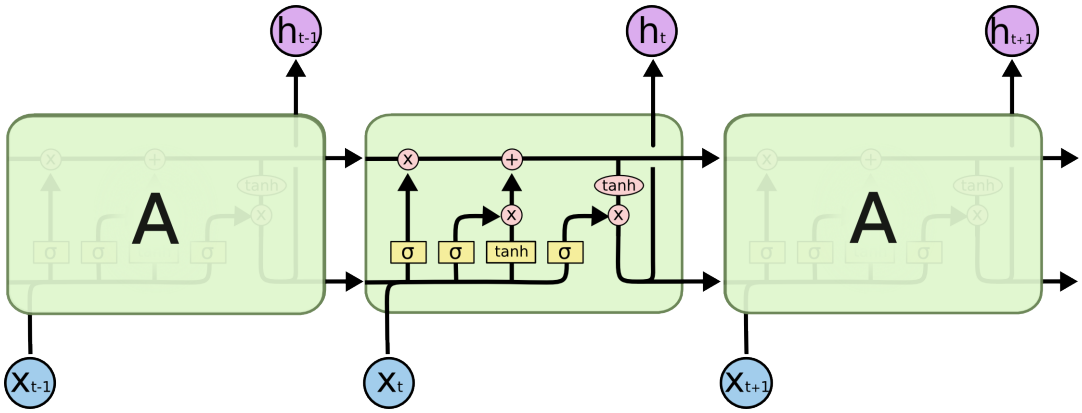


FIGURE 2.2 – Structure d'un réseau à mémoire court et long termes [1].

à jour de l'état caché afin de limiter le nombre de paramètres de la structure. Leurs performances sont comparables à celles des LSTM dont ils sont issus.

Les dernières avancées de traitement de données séquentielles, menées principalement dans le cadre du traitement automatique des langues est l'architecture d'attention appelée "Transformer" (VASWANI et al., 2017). Cette nouvelle architecture a rencontré un grand succès et les modèles s'appuyant sur ce principe continuent de montrer d'impressionants résultats (DEVLIN et al., 2018 ; BROWN et al., 2020). Le principe d'attention à la base de cette nouvelle architecture consiste à calculer pour chaque élément de la séquence quels sont les autres éléments pertinents. Il s'agit de déterminer pour chaque élément l'ensemble sur lequel il est pertinent de focaliser son attention pour réaliser la tâche souhaitée. Ainsi, si l'attention se focalise sur les éléments précédents de la séquence, le réseau a un comportement semblable à une LSTM présentée précédemment. Il est possible de voir les modèles récurrents comme des modèles focalisant leur attention uniquement sur les éléments précédents alors que le mécanisme d'attention proposé par l'architecture "Transformer" est beaucoup plus général. Contrairement aux réseaux récurrents cette nouvelle architecture n'a pas besoin de traiter les séquences dans un ordre précis puisqu'elle calcule la mise en relation des différents éléments de la séquence elle-même. Ce type d'architecture, représentée dans la figure 2.3, est plus complexe que les précédentes et est plus longue à entraîner.

2.3.3 Prioritisation pour l'apprentissage automatique

Cependant pour être capable d'entraîner ces réseaux, un grand nombre de données est nécessaire. Cela implique d'être capable d'enregistrer le système évoluant dans son environement. L'inefficacité des réseaux de neurones en terme de données impose la génération de grandes bases de données d'entraînement. Ces corpus sont souvent très déséquilibrés (SCHAAL, ATKESON et VIJAYAKUMAR, 2000) et l'entraînement est saturé par des exemples communs alors que les cas intéressants n'ont qu'un impact négligeable sur le procédé d'apprentissage. Enfin, le déséquilibre des corpus est un obstacle à l'apprentissage des cas difficiles.

La capacité des réseaux de neurones de continuer d'apprendre au fil de l'eau alors que de nouvelles données deviennent disponibles a été utilisée pour étendre progressivement les jeux de données vers les cas complexes (WILLIAMS et al., 2017b), offrant une solution à ce problème. Malheureusement cela implique de continuer d'entraîner les réseaux sur des cas bien connus alors que de nouvelles informations sont rarement

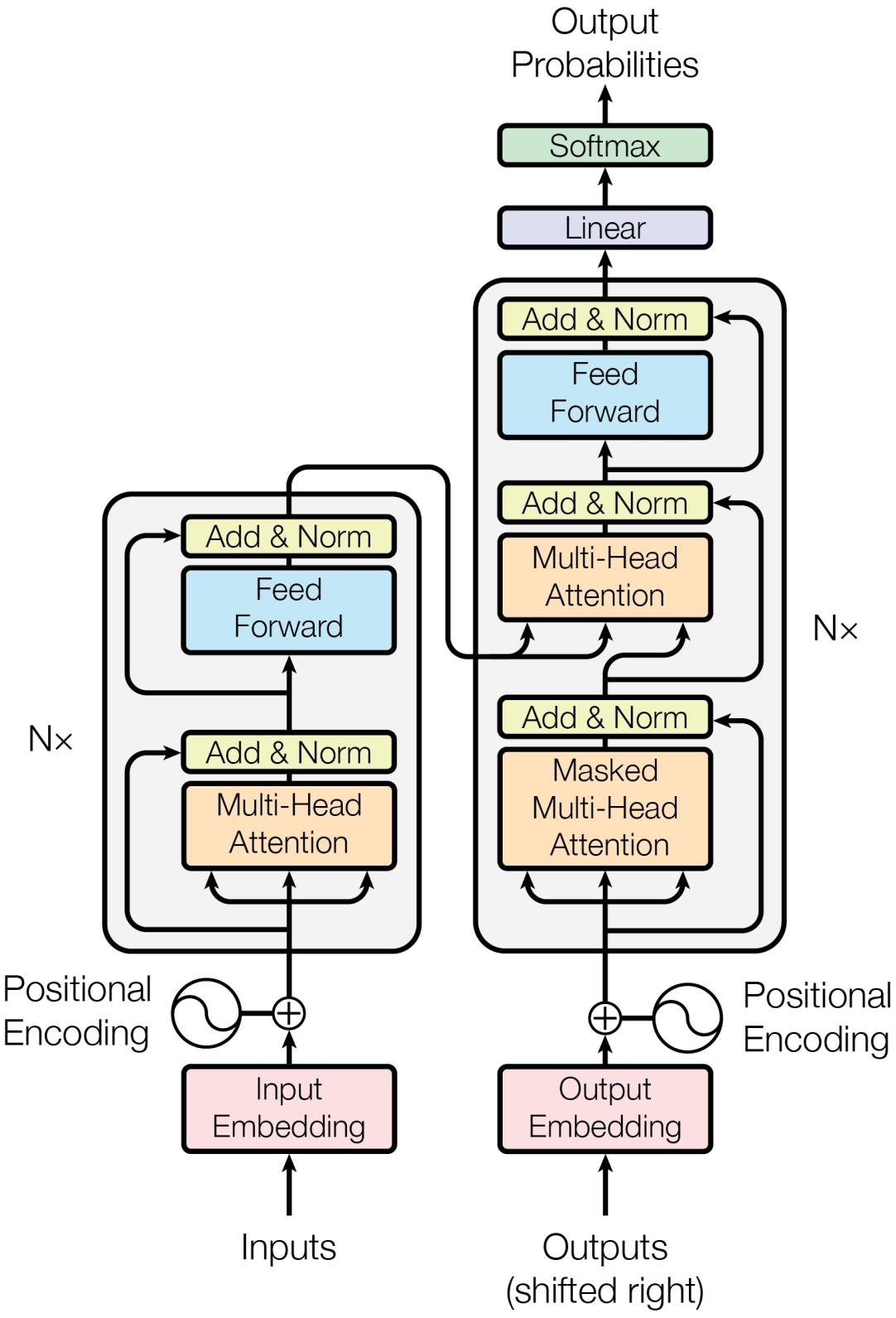


FIGURE 2.3 – Structure d'un réseau transformer, illustration issue de l'article : VASWANI et al., 2017.

disponibles. Identifier les exemples contenant plus d'informations afin de concentrer l'apprentissage dessus est une solution pertinente.

Le problème de la génération de données est bien connu lors de l'utilisation des méthodes d'apprentissage automatique. Par exemple, l'algorithme *Deep Q-Networks* (DQN) qui apprend à jouer à Atari (Mnih et al., 2013) a besoin de millions d'exemples pour être efficace. Une alternative intéressante pour résoudre ce problème a été développée dans Schaul et al., 2015 où les échantillons se voient attribuer une importance représentant la quantité d'informations qu'ils sont supposés apporter au processus d'apprentissage. Ceci est étroitement lié à l'échantillonnage préférentiel qui utilise aussi une pondération des échantillons (Katharopoulos et Fleuret, 2018).

Différentes méthodes d'entraînement et d'architectures de réseaux de neurones ont été présentées précédemment. La pertinence de l'utilisation de ces différents modèles en robotique implique de les évaluer au sein d'un algorithme de contrôle. Le chapitre suivant présente le détail d'un tel contrôleur, basé sur la modélisation de la dynamique du système.

Chapitre 3

Commandes prédictives

Afin d'accomplir une mission à l'aide d'un système robotique, il est nécessaire de lui communiquer une suite de commandes permettant d'atteindre les objectifs souhaités. Cela requiert la mise en place d'une stratégie permettant d'obtenir les commandes en fonction de l'état du système.

La stratégie considérée ici pour contrôler le système robotique est la commande prédictive qui a démontré de nombreux avantages dans le cadre du contrôle de robots mobiles (Lenain et al., 2006 ; Lucet, Lenain et Grand, 2015), dont des drones (Naegeli et al., 2017 ; Dentler et al., 2016 ; Zhang et al., 2015).

L'idée de la commande prédictive est d'utiliser un modèle de la dynamique du système afin de prédire le résultat des commandes appliquées à ce dernier. Cette capacité d'anticipation permet de sélectionner les commandes en fonction de l'évolution de l'état du système qu'elles provoquent.

Une fois les prédictions faites, le choix des commandes à envoyer effectivement aux actionneurs est arbitré par la qualité du résultat au regard de la tâche à compléter. Un exemple typique dans le cas de la commande d'un drone est de considérer une commande qui rapproche de la cible comme de bonne qualité alors qu'une commande provoquant une collision avec un arbre sera de très mauvaise qualité. La formalisation de cette notion se fait grâce à une fonction de coût qui permet de décider quelles sont les commandes les plus adaptées. La sélection de la trajectoire dans l'espace des commandes se fait en minimisant cette fonction.

3.1 Fonction de coût

Afin d'évaluer la qualité d'une trajectoire, il est nécessaire de définir une fonction de coût. Soit $X(t) \in \mathbf{X}$ l'état du système avec $\mathbf{X} \subset \mathbb{R}^n$ l'ensemble des états possibles du système, en considérant un état de dimention n. Soit $U(t) \in \mathbf{U}$ la commande du système avec $\mathbf{U} \subset \mathbb{R}^m$ l'ensemble des commandes possibles du système, en considérant que la dimension de la commande est m. Soit la fonction de coût $\phi : \mathbf{X} \times \mathbf{U} \longrightarrow \mathbb{R}^+$. Le but de cette fonction est de codifier les objectifs de la mission. Il peut s'agir, par exemple, de suivre une cible ou une piste, ou de passer par un point précis. Les contraintes telles que ne pas s'approcher des obstacles, respecter des normes de sécurité, minimiser l'énergie dépensée ou le temps pris pour réaliser l'objectif sont aussi des éléments pris en compte. L'objectif de notre algorithme de contrôle est de trouver la suite de commandes permettant de minimiser cette fonction comme décrit dans l'équation 3.1.

$$U^* = \arg\min_{U \in \mathbf{U}} \sum_{t=t_i}^{t_f} \phi(X(t), U(t)) \tag{3.1}$$

De façon équivalente, il est possible de considérer une fonction de récompense à maximiser. Dans la suite, pour des raisons de clarté et de cohérence, il est choisi de considérer une fonction de coût à minimiser.

3.1.1 Carte de coût

Représenter une piste ou un obstacle revient à donner un coût à tout point de l'espace. Cela peut être décrit sous la forme d'une carte de coût. Il s'agit d'associer tout point de l'espace à un coût qui reflète la distance à l'objectif et les contraintes spatiales comme les obstacles.

La figure 3.1 montre un exemple simple de carte de coût : la partie en gris représente le centre de la piste, pour lequel le coût est nul, la partie en bleu les bords de la piste avec un coût moyen et en rose l'extérieur de la piste avec un coût très élevé.

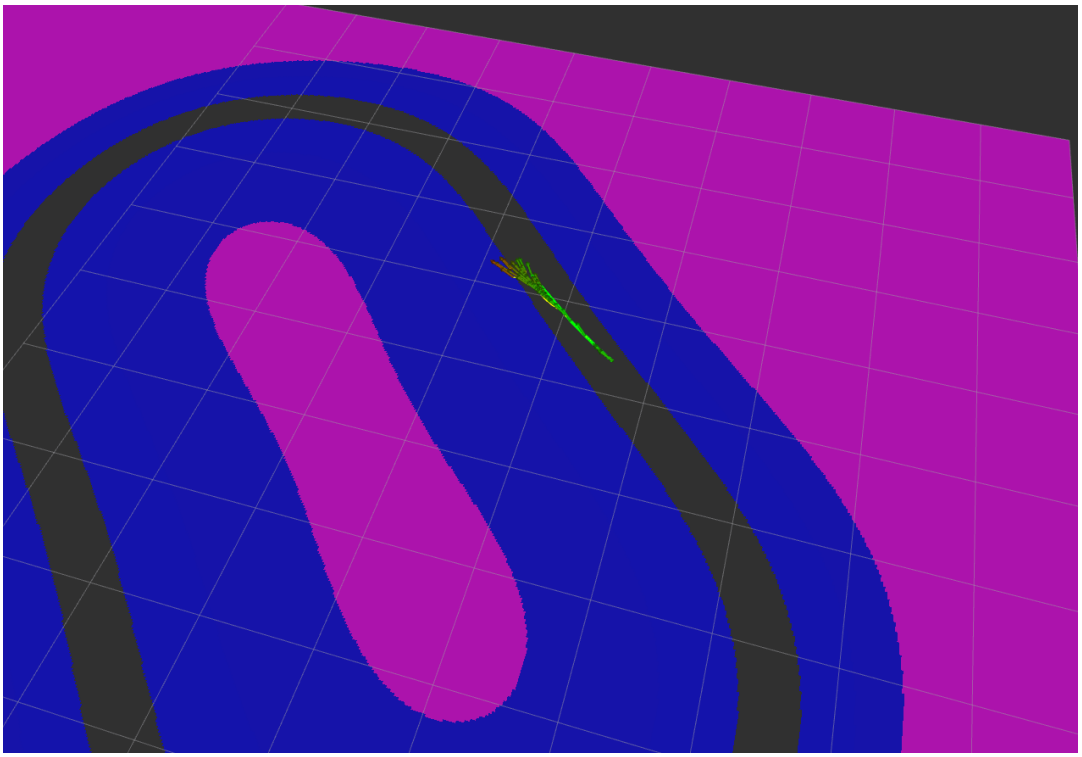


FIGURE 3.1 – Exemple de carte de coût.

Une carte de coût peut être fixe : il faut rester sur la piste, la piste est connue dans l'absolu. L'ensemble des points acceptables pour le robot est alors établi. Il est aussi possible de considérer des cartes dynamiques, où le robot évalue l'espace dans son entourage grâce à ses capteurs. C'est, par exemple, le cas d'une carte traquant les obstacles détectés grâce à un laser porté par le robot. Dans ce cas, la carte est relative à la position du robot et évolue en fonction des données récoltées par les capteurs.

Un certain nombre d'écueuils sont à éviter dans la conception d'une carte de coût. Tout d'abord une carte de coût excessivement punitive peut mener à des situations impossibles. Par exemple, la figure 3.2 montre le résultat du test suivant : suivre une piste carrée avec un bateau. Afin de garantir que le bateau reste le plus au centre possible de la piste, un coût important est donné à tout l'espace en dehors du milieu de la piste. Dans ces conditions, le bateau se retrouve coincer au premier angle car l'algorithme ne peut pas trouver de trajectoire satisfaisante et le minimum de coût se stabilise sur la trajectoire correspondant à un arrêt du bateau.

En effet, la composante de la fonction de coût encourageant le bateau à se déplacer ne peut pas fonctionner ici. Dans le cas présenté dans la figure, les trajectoires testées avec une vitesse préférable doivent couper l'angle. Or cela est extrêmement pénalisé par la carte de coût et donc, ces trajectoires ne sont pas retenues. Avec une telle topologie pour la carte de coût, il ne suffit pas d'augmenter la composante liée à la vitesse pour motiver le bateau à bouger. Le problème inverse apparaîtrait alors : toute trajectoire avec une vitesse souhaitée sera considérée valide même si elle sort de la piste. Pour pouvoir concilier ces deux éléments, il est nécessaire de considérer un couloir autour de la piste pour lequel le coût reste très faible de sorte que le bateau ait un minimum de marges de manœuvre.

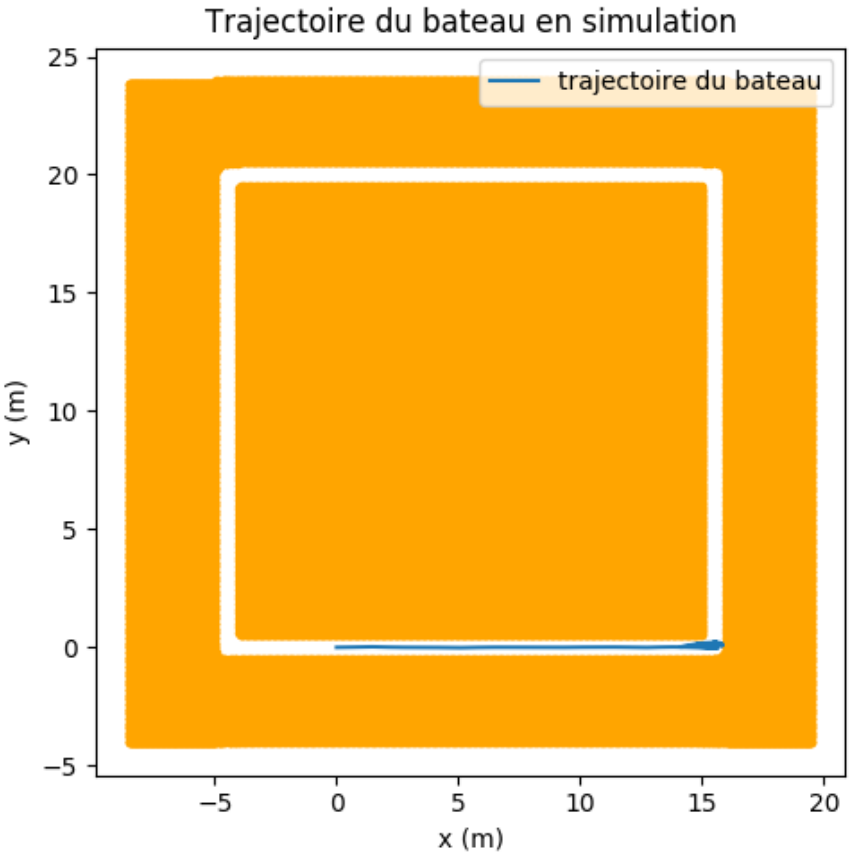


FIGURE 3.2 – Problème d'une carte de coût trop sévère.

Un autre aspect à garder à l'esprit lors de la conception d'une carte de coût est la présence d'informations. Intuitivement, il semble pertinent de penser que le centre de la piste est préférable au bord de la piste. Cependant, si la piste est implémentée comme un ensemble de points ayant un coût nul comme dans la figure 3.3, aucune information ne permet à l'algorithme de différencier un point proche du bord de la

piste d'un point au centre. Dans ce cas, le bateau se retrouve au bord de la piste, ce qui le met en difficulté rapidement. Il est aisé de résoudre le problème en rajoutant un faible coût proportionel à la distance au centre de la piste.

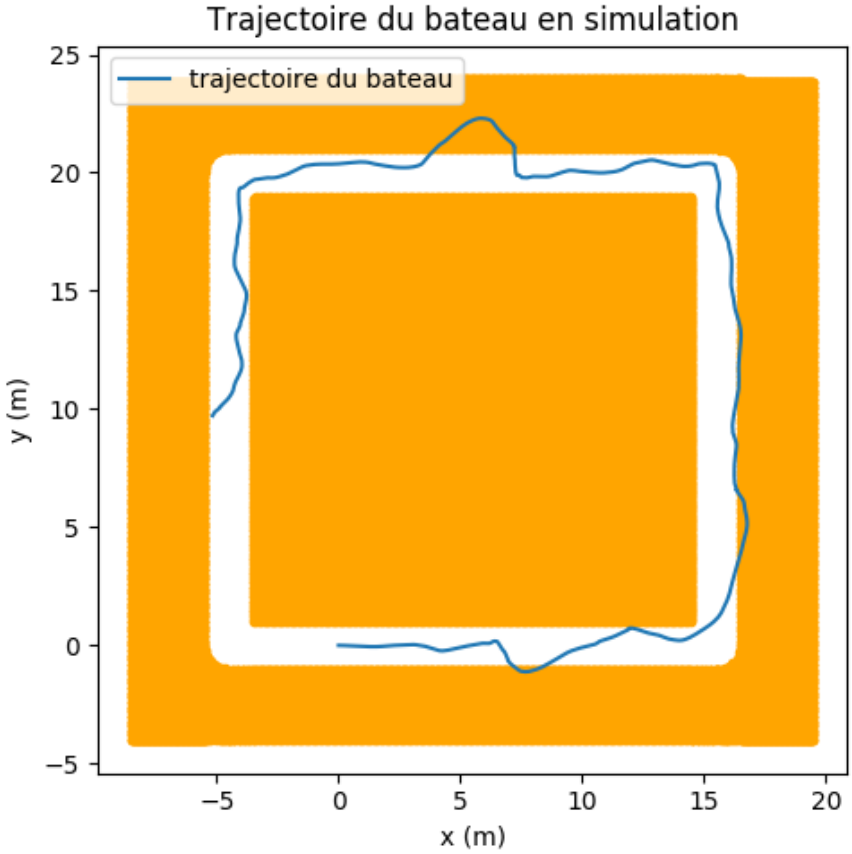


FIGURE 3.3 – Exemple du problème d'une carte avec un manque d'information : aucune information n'est donnée sur le centre de la piste.

3.1.2 Autres éléments pour la fonction de coût

Il est souvent intéressant de considérer des coûts qui ne sont pas directement liés à la position du système. Par exemple, les tâches d'inspection requièrent souvent le maintien d'une vitesse prédéterminée. Il est donc naturel d'inclure une composante liée à la vitesse dans la fonction de coût. Typiquement, un coût proportionnel à la différence entre la vitesse actuelle du robot et la vitesse souhaitée. D'autre part, il est aussi possible d'inclure des contraintes de sécurité telles que des limitations sur les accélérations maximales.

Plus une fonction de coût est composée de nombreux paramètres, plus il est difficile d'équilibrer les différentes composantes et d'en évaluer leurs effets. L'interaction entre différents paramètres peut aussi faire apparaitre des situations où la fonction de coût décourage l'achèvement de la tâche. Un tel cas est arrivé lors des expérimentations en simulation. La carte de coût seule ne donne pas de raison de remplir la tâche "parcourir la piste" car s'arrêter au milieu de la piste permet de minimiser le coût. Cependant l'ajout d'une composante de coût trop élevée sur la vitesse inhibe le coût de la carte et le bateau ne respecte plus les limites de la piste.

Afin de concevoir une fonction de coût pertinente pour une tâche donnée, il est nécessaire d'inclure les différentes composantes proportionnellement à leur impact sur la qualité du résultat. Pour s'assurer qu'elles ne seront jamais franchies, les contraintes de sécurité peuvent être incorporées avec des coûts très supérieurs aux autres. Les contraintes fondamentalement liées à la tâche comme le tracé de la piste doivent être prépondérantes. Enfin, les contraintes informatives quant à la nature de la tâche, à remplir une fois que les autres sont assurées, doivent garder un coût plus faible.

3.1.3 Suivi de cible : une alternative à la carte de coût

Il est parfois plus intéressant de considérer une trajectoire cible à suivre : cela permet notamment d'incorporer facilement une notion de vitesse. La trajectoire cible est définie comme la suite des positions à atteindre en un temps souhaité. Il est possible de considérer l'utilisation d'une carte de coût qui changerait à chaque pas de temps. Cependant, cela ne serait pas très efficace et il est préfèrable d'exprimer l'erreur comme une distance entre l'état du robot à un instant donné et la cible correspondante. Les contraintes en vitesse sont naturellement incorporées via l'aspect temporel des positions au sein de la trajectoire.

Dans les expériences effectuées avec le drone, la fonction de coût s'exprime de cette façon : une trajectoire cible est publiée en continu et permet un calcul efficace de la fonction de coût par mesure de la distance entre la position du drone et la position de la cible à chaque instant.

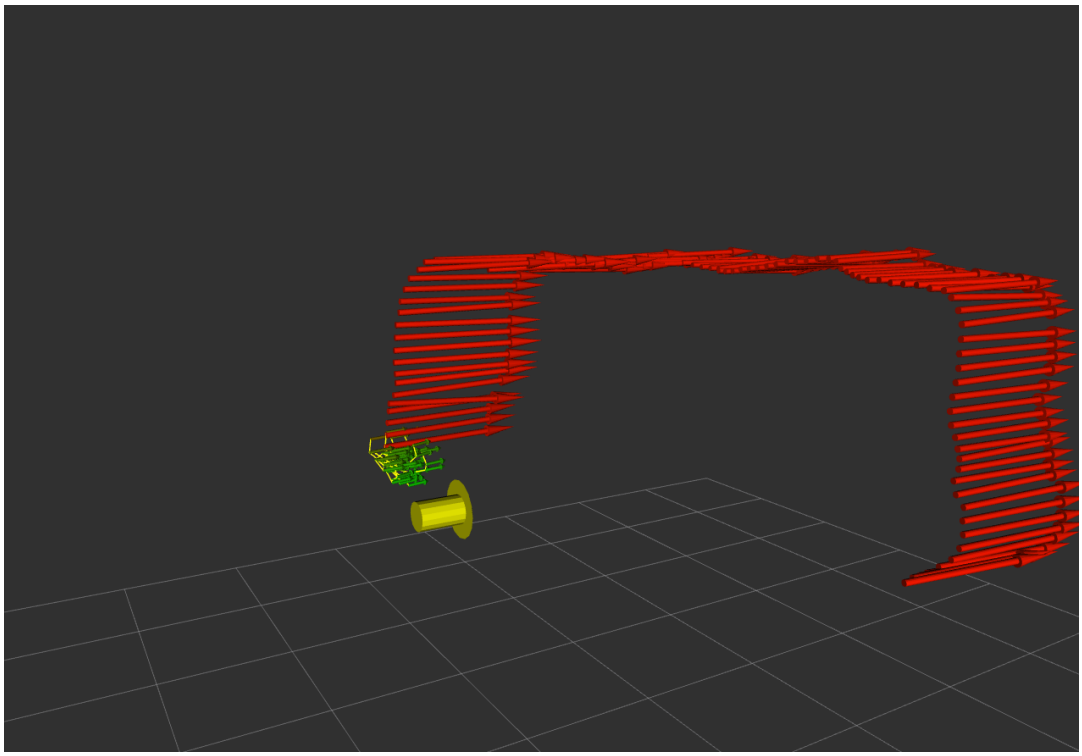


FIGURE 3.4 – Suivi de cible avec un drone en simulation

Dans la figure 3.4, la cible est en jaune, la trajectoire parcourue par le drone en rouge et les trajectoires évaluées par le contrôleur en vert.

3.2 Le rôle du modèle

La qualité de la commande est une conséquence directe de la précision des prédictions. La fonction que l'on cherche à modéliser est la fonction F de l'évolution du système. Cette fonction calcule l'état suivant du système x_{t+dt} étant donné l'état x_t et la commande u_t. Il est possible de considérer un historique des états précédents de même qu'un historique des commandes. F est décrite dans l'équation 3.2.

$$x_{t+dt} = F(x_t, u_t). \tag{3.2}$$

Plus spécifiquement, lorsque l'étude porte sur des robots mobiles, l'objectif est d'identifier la dynamique du modèle, l'état du système correspondant alors aux positions qui sont obtenues par intégration des vitesses comme montré dans l'équation 3.3.

$$\begin{cases} p_{t+1} = p_t + v_t dt \\ v_{t+1} = v_t + f(v_t, u_t) dt \end{cases} \tag{3.3}$$

En effet, dans l'équation 3.1 précédente, les états futurs $x(t), t > t_i$, qui correspondent aux positions p_t dans l'équation 3.3, ne peuvent être connus que par l'intermédiaire du modèle. Dans le cas d'une précision parfaite, c'est-à-dire capable de prédire le résultat de n'importe quelle commande avec exactitude, il est possible de prévoir la trajectoire voulue à l'avance avec précision. Si un modèle linéarisable est connu et étant donnée une fonction de coût, alors il est possible de résoudre le problème d'optimisation (Rawlings et Mayne, 2009) et de trouver le meilleur ensemble de commandes possibles.

À l'inverse, un modèle erroné ne permet de ne tirer aucune conclusion et la prise de décisions est hasardeuse. En général, les modèles sont considérés fiables sur un certain horizon, c'est-à-dire pour un certain nombre de pas de temps discrets. Plus le modèle est performant, plus il est utilisable sur un horizon long et meilleure sera la capacité de l'algorithme à anticiper. L'impact de l'utilisation des différents modèles sera étudié dans le chapitre suivant.

Les systèmes considérés ici utilisent des contrôleurs à commandes numériques. Il est donc nécessaire de discrétiser le temps. Le pas de temps peut être determiné par des contraintes de capteurs, de temps de calcul, par les caractéristiques du système, par les données disponibles ou par les objectifs de la mission. Plus le pas de temps est faible, plus il est facile d'obtenir des modèles performants, mais plus les contraintes sont fortes en terme de temps de calcul. D'autre part, les erreurs s'accumulant avec les pas de temps, il peut être dommageable d'avoir une période très faible lorsque l'horizon des prédictions est fixe. En effet, l'accumulation de petites erreurs peut mener à une erreur finale plus grande qu'une erreur moyenne sur un pas de temps large.

La commande prédictive a été décrite comme le calcul de commandes optimales pour un modèle et une fonction de coût donnés. Après avoir discuté de la fonction de coût et de la place du modèle, la section suivante s'intéresse au détail du calcul des commandes.

3.3 Utilisation de l'intégrale de parcours

Étant donnés un modèle et une fonction de coût, il existe différentes façons de calculer les commandes recherchées. En général, la méthode privilégiée consiste à optimiser en ligne la fonction de coût. Cette optimisation s'appuie sur des méthodes de type Gauss-Newton tel quel *Iterative Linear Quadratic Regulator* (ILQR) (Tassa, Erez et Todorov, 2012). Ce type de méthode contraint la fonction de coût (qui doit être convexe) et requiert une approximation au premier ou au second ordre de la dynamique. Ici, une version plus flexible de la commande prédictive est utilisée : la commande prédictive à intégrale de parcours. Elle présente l'intérêt de pouvoir fonctionner avec une plus grande variété de fonctions de coût sans nécessité d'approximation au premier ou second ordre de la dynamique. Cette méthode est détaillée dans ce qui suit.

De façon générale, la méthode consiste à échantillonner un grand nombre de trajectoires dans l'espace des commandes, à simuler les trajectoires correspondantes pour l'état du système, à les évaluer grâce à une fonction de coût et finalement à déduire de cette évaluation la séquence de commandes optimale à envoyer au système. La première commande de cette séquence est appliquée puis le processus recommence de façon itérative.

La taille des trajectoires échantillonnées dans l'espace des commandes dépend de l'horizon choisi. Plus l'horizon est lointain, meilleure est la capacité d'anticipation de l'algorithme. Cependant, plusieurs facteurs viennent limiter la longueur de la trajectoire. En effet, plus la trajectoire est longue, plus il y a d'étapes dans la trajectoire. Et à chaque étape, le modèle calcule l'étape suivante. Les erreurs de modélisation s'acumulent donc au cours de la trajectoire. Plus l'horizon est lointain, plus il y a d'incertitude sur l'état du système en fin de course. De plus, l'espace de recherche est agrandi. En effet, l'ensemble des trajectoires possibles dans l'espace des commandes augmente exponentiellement avec le nombre d'éléments dans la trajectoire. Plus il y a d'étapes à calculer au sein d'une trajectoire, plus le coût en temps de calcul de cette trajectoire est élevé. Cela contraint d'autant plus le nombre d'étapes. Le compromis qui s'impose sera discuté expérimentalement dans le paragraphe 3.4.2.

En robotique, la plupart des espaces de commande sont continus. C'est notamment le cas d'un drone commandé en vitesse. L'espace des commandes en vitesse du drone est continu. Il en résulte une infinité de commandes possibles. Lorsque cet espace est échantilloné pour trouver les commandes optimales recherchées, plus le nombre d'échantillons est grand, plus il est facile d'explorer l'espace des commandes. Cependant, plus le nombre d'échantillons est grand, plus le calcul est coûteux. Afin de maximiser la pertinence des échantillons choisis, la méthode par l'intégrale de parcours concentre l'échantillonnage autour d'une commande initiale, puis utilise la trajectoire calculée à une étape comme trajectoire initiale de l'étape suivante. Ainsi la recherche progresse autour d'une trajectoire de plus en plus pertinente. Il est montré dans Williams et al., 2017b que cela permet de converger vers la trajectoire des commandes optimale.

D'autre part, l'échantillonnage des commandes n'est pas nécessairement entièrement laissé au hasard. En effet, la distribution choisie pour échantillonner les commandes est de grande importance. Le plus naturel est d'utiliser une distribution normale dont la variance est choisie. Plus la variance est grande, plus l'espace des commandes sera

exploré. Cela peut porter préjudice à l'efficacité de l'algorithme si la structure du problème implique que la plupart des commandes sont très mauvaises et que seules certaines sont intéressantes. Ce cas est en fait souvent rencontré. C'est le cas d'une fonction de coût avec une piste ou un point de passage par exemple. Dans un tel cas, il peut être plus intéressant d'échantillonner les commandes avec une faible variance. Il s'agit ici de concentrer la recherche dans un sous-espace des commandes approprié.

Il y a plusieurs méthodes pour choisir la commande à envoyer étant donnés le résultat de la prédiction et le coût associé. Par exemple, une approche consiste à évaluer l'ensemble des trajectoires calculées et de prendre la meilleure. Cette approche intuitive n'est pas nécessairement la plus efficace car elle ne profite pas de l'ensemble des informations que fournissent les prédictions. En effet, cela revient à dire que seule la meilleure prédiction donne des informations pertinentes. Cependant le fait qu'une trajectoire proche de celle-ci ait un coût très important donne des informations exploitables. Soit le cas d'un obstacle et de deux trajectoires qui ne diffèrent que par leur dernière commande : l'une envoie le robot dans l'obstacle, l'autre pas. La trajectoire rencontrant l'obstacle aura un coût rédibitoire alors que l'autre aura un coût très favorable. Pourtant, les premières commandes, communes aux deux trajectoires, vont toutes les deux envoyer le robot dangereusement proche de l'obstacle. Or il est possible d'utiliser l'information de la trajectoire ayant un coût élevé pour diminuer la confiance dans l'autre trajectoire. De même la confiance dans la trajectoire en échec est augmentée par sa proximité avec une trajectoire de bonne qualité.

C'est l'intérêt de la méthode des intégrales de parcours qui calcule la suite de commandes à envoyer au système comme une moyenne pondérée des commandes testées par la simulation. Chaque commande est affectée d'un poids correspondant à la valeur totale de la trajectoire dont elle fait partie.

Dans la suite, le contrôleur à commande prédictive décrit dans l'algorithme 1 [1] est utilisé.

Les éléments importants sont les suivants :

— à chaque étape de l'algorithme, des variations, notées ici ϵ, appliquées sur la commande sont échantillonées,
— ensuite, le modèle dynamique F de notre système est utilisé pour prédire l'évolution des différentes trajectoires,
— une fois ces trajectoires calculées, elles sont évaluées à l'aide de la fonction de coût ϕ,
— le résultat de cette évaluation permet de calculer les coefficients w_k,
— ensuite, ils sont utilisés pour combiner l'ensemble des commandes échantillonnées en une suite de commandes U,
— la première commande de cette suite est appliquée au système tandis que les suivantes servent à l'initialisation de l'étape suivante.

3.4 Application au contrôle

Afin d'évaluer la pertinence de l'algorithme de contrôle choisi, il est testé sur un système en simulation. La simulation offre une grande flexibilité ainsi qu'une facilité de mise en place qui permet de rapidement faire de nombreux tests. L'objectif est de déterminer la capacité de la commande prédictive à parcours d'intégrale à effectuer une

1. Algorithme décrit par Williams et al., 2017a simplifié pour une température neutre.

Algorithme 1 Commande prédictive à parcours d'intégrale

Entrée: F : Modèle dynamique
T : Nombre d'étapes
K : Nombre de trajectoires échantillonnées
ϕ : Fonction de coût
u_t : Commande calculée pour l'instant t
s_t : État à l'instant t
$U = \boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_T$: séquence initiale des commandes

Échantilonage de $\epsilon_k = \epsilon_k^1, \epsilon_k^2, \ldots, \epsilon_k^T \sim \mathcal{N}(\mu, \sigma^2)$
pour $k = 0$ à $K - 1$ **faire**
  pour $t = 1$ à T **faire**
    $u_t = \boldsymbol{u}_t + \epsilon_k^t$
    $s_{t+dt} \leftarrow F(s_t, u_t)$
  fin pour
  $S_k = \{s_t \ \forall t \in [0, T]\}$
  $C_k \leftarrow \phi(S_k)$
fin pour
$\beta \leftarrow min_k[C_k]$

$\eta \leftarrow \sum_{k=0}^{K-1} exp(-(C_k - \beta))$

pour $k = 0$ to $K - 1$ **faire**
  $w_k \leftarrow \frac{1}{\eta} exp(-(C_k - \beta))$
fin pour
pour $t = 1$ à T **faire**
  $\boldsymbol{u}_t = \boldsymbol{u}_t + \sum_{k=1}^{K} w_k \epsilon_k^t$
fin pour
retourne U

tâche de suivi de piste en simulation, ainsi que d'étudier les différents paramètres de l'algorithme. Dans un premier temps, l'impact de l'horizon des trajectoires testées par l'algorithme est étudié. Dans un second temps, l'influence du nombre d'échantillons utilisés est analysée. Et enfin dans un troisième temps, l'importance de la variance de l'échantillonnage des commandes est discutée.

3.4.1 Contexte expérimental

Les tests sont faits sur la plateforme *Héron*, un véhicule autonome de surface (USV) qui permet de facilement mettre en avant les effets des différents paramètres du fait de son inertie et de la complexité de sa dynamique. L'implémentation du contrôleur à commandes prédictives décrit précédemment est faite en python sous la forme d'un paquet ROS. Une attention particulière est donnée à l'écriture de la recherche Monte-Carlo afin de maximiser l'éfficacité des calculs. En effet ces calculs coûteux doivent être entièrement réalisés dans l'intervalle de 200 ms afin de respecter la fréquence de contrôle de 5 Hz. La majorité des calculs matriciels est faite à l'aide de la bibliothèque *numpy*. L'objectif de l'utilisation de cette bibliothèque est d'utiliser l'efficacité, proche des performances C++, de ses fonctions. D'autre part le modèle utilisé pour les résultats présenté ici est un modèle neuronal, dont le détail de la structure et de l'entraînement sont détaillés dans le chapitre suivant. L'implémentation de ce réseau est faite à l'aide de TensorFlow et fonctionne sur carte graphique.

La tâche sur laquelle l'évaluation est faite consiste au suivi d'une piste carrée. Les angles étant la difficulté qui permet d'observer facilement les différences de comportement suivant les paramètres. La carte de coût utilisée pour les expériences suivantes est montrée dans la figure 3.5.

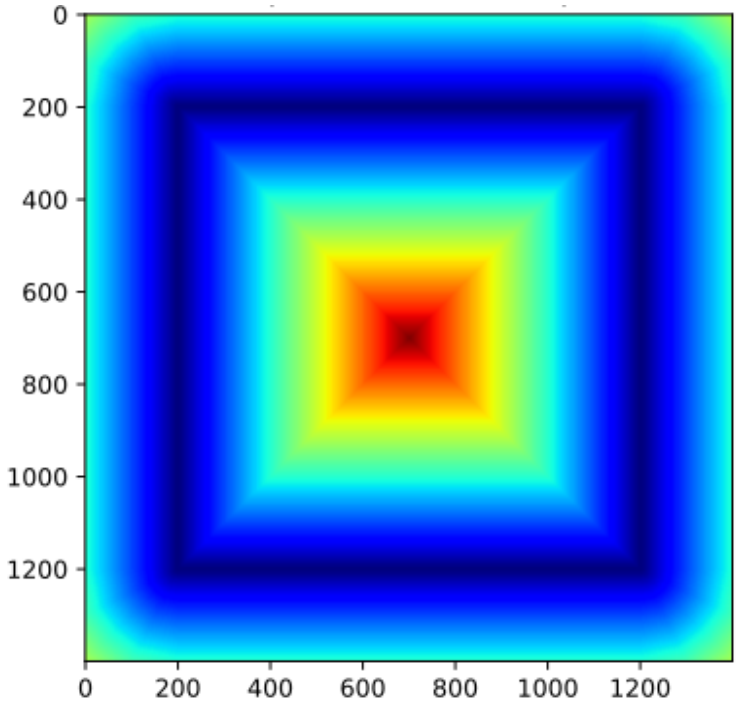


FIGURE 3.5 – Carte de coût utilisée pour les expériences de suivi de la piste carrée.

Pour l'environnement de simulation, le logiciel Gazebo est utilisé. La structure logicielle ROS est mise à profit pour l'interaction et l'intégration des différents algorithmes impliqués dans les expériences. Pour le bateau, le packet "Heron" développé par Clearpath Robotics est utilisé. Le packet "UUV simulator" est utilisé pour le réalisme de la modélisation des non linéarités des turbines ainsi qu'une description avancée de la fluidité de l'eau.

Les résultats suivants s'intéressent à la robustesse du contrôle prédictif en fonction des principaux paramètres de l'algorithme. Il s'agit d'évaluer la répétabilité des résultats obtenus. Pour cela les performances moyennes en terme de positions, c'est à dire de coût de carte, ainsi qu'en terme de vitesses sont évaluées. Les moyennes sont faites sur 15 essais. Les paramètres évalués sont l'horizon de la trajectoire, le nombre d'échantillons de commandes évaluées par l'algorithme et la variance de l'échantillonnage des commandes.

3.4.2 Influence de l'horizon de la trajectoire

L'horizon de la trajectoire détermine à quel point le contrôleur tente de prédire à long terme. Un nombre trop faible de pas pour la trajectoire la rendra trop courte pour que l'algorithme puisse prendre en compte la dynamique du bateau. Il ne sera pas capable d'anticiper assez les angles pour pouvoir tourner correctement. À l'inverse, en regardant trop loin dans le futur, le modèle cumule ses erreurs. En effet, chaque prédiction étant construite sur les résultats précédents, l'erreur s'amplifie à chaque étape.

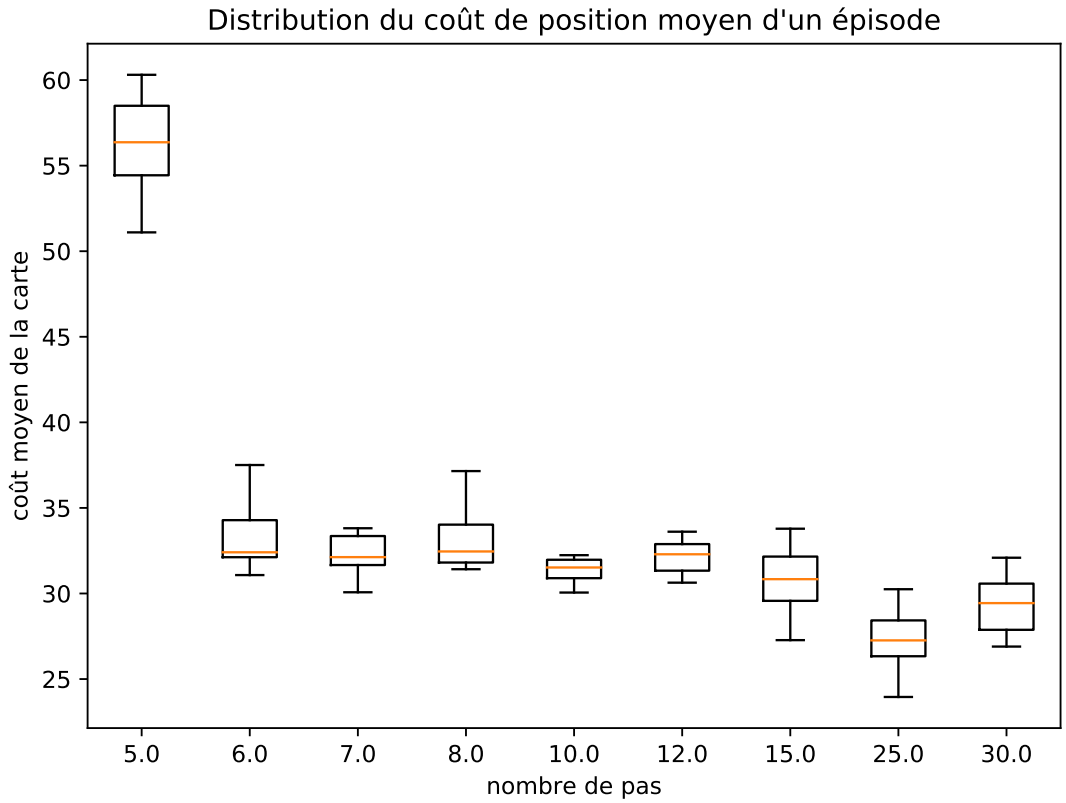


FIGURE 3.6 – Résultats moyens du suivi de piste en fonction de l'horizon des trajectoires pour un bateau en simulation.

La figure 3.6 montre l'influence de la longueur de la trajectoire utilisée par le contrôleur prédictif. Dans ce graphique, le nombre de commandes échantillonnées est fixé à 1500 et l'écart-type de cet échantillonnage à 0,75. En dessous de 5 pas, les trajectoires sont tellement courtes qu'elles ne peuvent pas prendre en compte la dynamique du bateau et le contrôleur est inutile. À partir de 40 pas, le coût en calcul devient trop important et il n'est plus possible de calculer de façon régulière les commandes pour le système. En conséquence, les performances sont très dégradées et le contrôleur ne fonctionne plus. La figure 3.6 montre que pour un nombre de pas égal à 5, le contrôleur est en grande difficulté pour suivre la carte. En effet il n'est pas capable d'anticiper correctement. L'augmentation du nombre de pas améliore les performances jusqu'à une trajectoire d'une longueur de 25 pas. Après quoi, les résultats se dégradent légèrement. Cela peut être causé par le fait que, lorsque la trajectoire devient plus longue, l'erreur de position augmente comme expliqué précédemment. Une autre hypothèse est que l'augmentation de la longueur des trajectoires nécessite une augmentation du nombre d'échantillons afin que l'exploration des trajectoires soit pertinente.

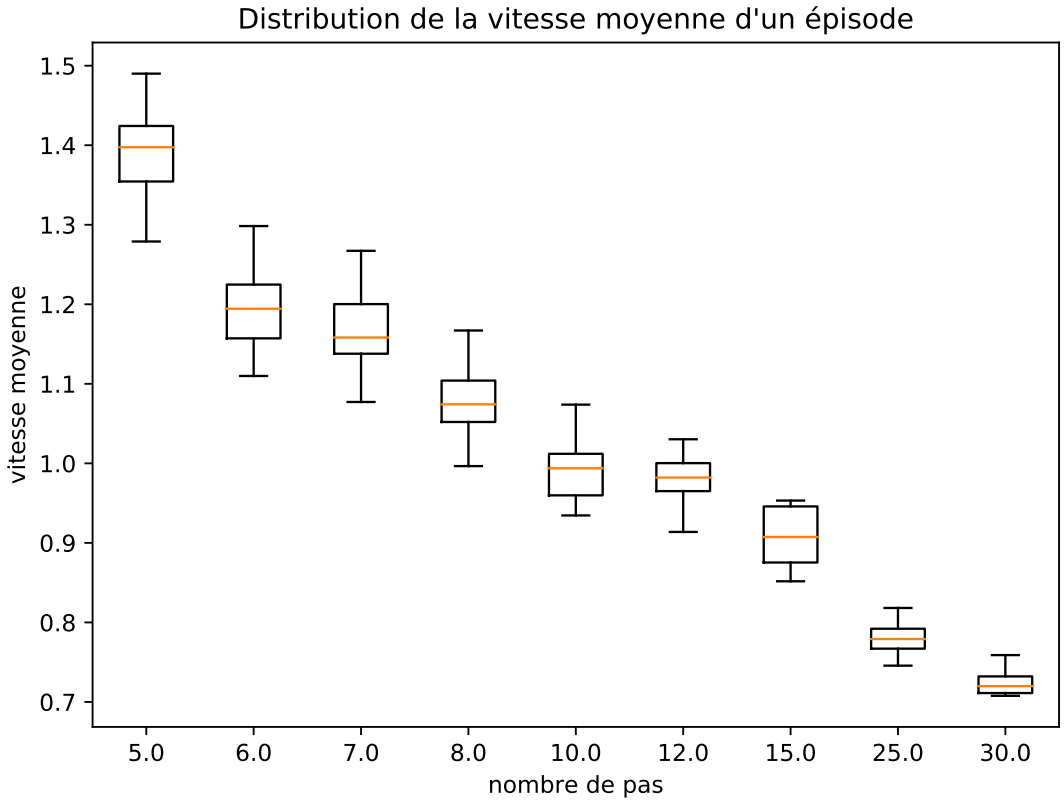


FIGURE 3.7 – Vitesses moyennes en fonction de l'horizon des trajectoires pour un bateau en simulation.

Un autre critère d'évaluation pour le contrôleur est la vitesse moyenne avec laquelle il parcourt le circuit. En effet, s'il respecte correctement la piste, une vitesse supérieure est préférable. On contate dans la figure 3.7 qu'à mesure que le nombre de pas augmente, la vitesse diminue. En effet, plus les trajectoires sont longues, plus les vitesses élevées risquent d'être pénalisantes d'un point de vue du coût de la carte. Un nombre de pas plus élevé favorise donc un meilleur respect de la piste, comme vu dans la figure 3.6, aux dépens des vitesses.

3.4.3 Influence du nombre d'échantillons

Un autre paramètre influant de manière importante les résultats du contrôleur est le nombre de trajectoires échantillonnées par l'algorithme. Un nombre de trajectoires trop faible signifie qu'on ne couvrira pas suffisamment l'espace des actions pour obtenir assez d'informations. À l'inverse, un grand nombre de trajectoires permet de tester tous les cas intéressants. Cependant, une grande quantité d'échantillons augmente rapidement le coût des calculs.

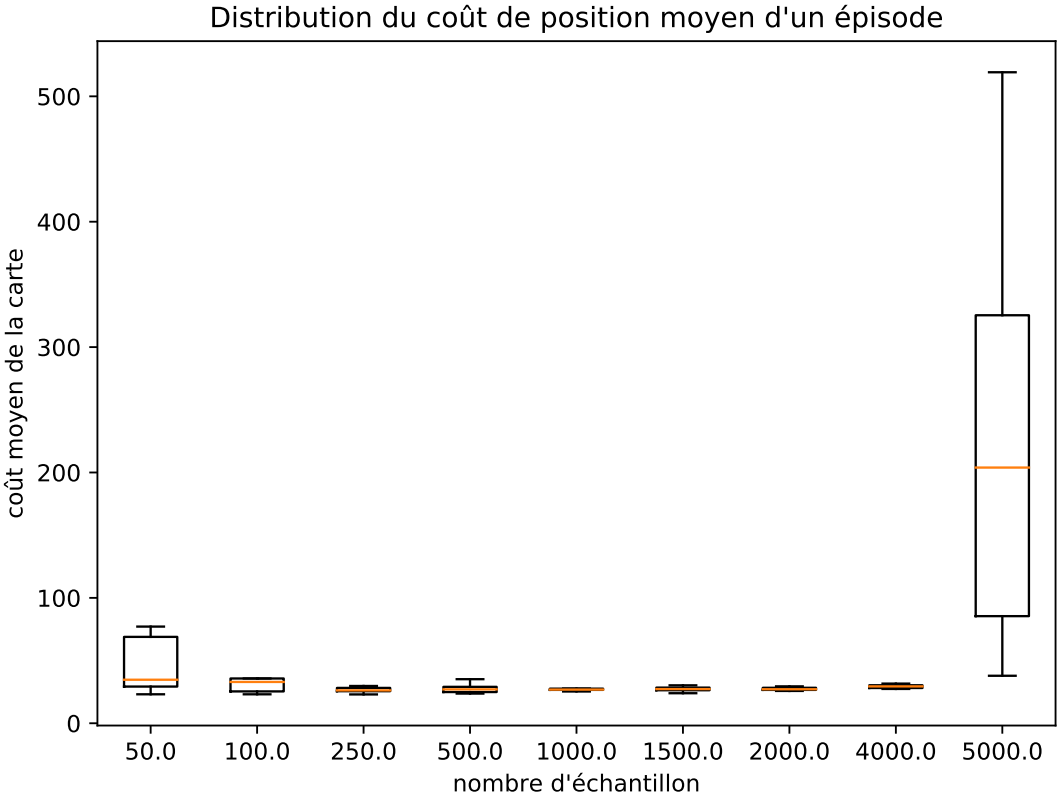


FIGURE 3.8 – Résultats moyens du suivi de piste en fonction du nombres d'échantillons pour un bateau en simulation.

La figure 3.8 décrit la moyenne du coût de la carte en fonction du nombre d'échantillons. Pour ce graphique le nombre d'étapes est fixé à 25, et l'écart-type de l'échantillonnage est de 0,75. Pour un faible nombre d'échantillons (50), les performances sont médiocres. À l'inverse, pour un très grand nombre d'échantillons (5000), le contrôleur étant à la limite des capacités de calcul disponible, certains essais sont des échecs complets. En effet, à partir de ce nombre d'échantillons, il n'est plus possible pour le contrôleur de fonctionner en temps réel, il y a donc un délai entre les observations et le calcul de la trajectoire optimale. Dans l'ensemble, la moyenne du coût moyen diminue lorsqu'on augmente le nombre d'échantillons jusqu'à ce que le coût en calcul devienne le facteur limitant. D'après le coût en position, les meilleurs résultats sont obtenus pour un nombre d'échantillons entre 1000 et 4000.

Les résultats observés sur le coût moyen de la carte se retrouvent en terme de vitesse. La figure 3.9 montre que, pour un nombre d'échantillons supérieur à 4000, les performances en vitesse sont dégradées par le problème du coût des calculs. Avant de

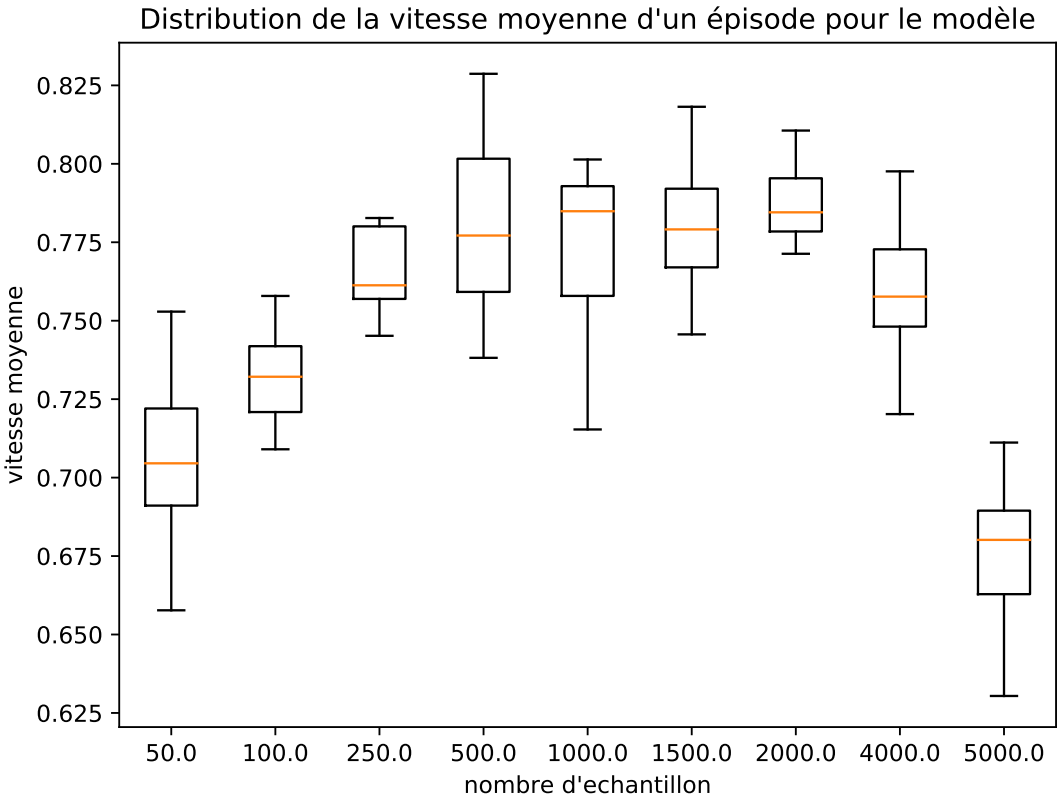


FIGURE 3.9 – Vitesses moyennes en fonction du nombre d'échantillons pour un bateau en simulation.

rencontrer cette limite, l'augmentation du nombre d'échantillons améliore les performances. Les résultats sont particulièrement mauvais pour un nombre d'échantillons faible (50-100). Au-delà, l'augmentation du nombre d'échantillons augmente faiblement les performances en vitesse jusqu'à un nombre d'échantillons égal à 2000.

3.4.4 Influence de l'écart-type de l'échantillonnage

La variance de l'échantillonnage permet de choisir à quel point l'espace des commandes est exploré. Plus la variance est grande, plus il y a d'explorations. Mais cela implique aussi qu'il y a plus de bruit donc plus de trajectoires inintéressantes sont échantillonnées. Pour une faible variance, le système n'explore pas assez et trouve des trajectoires qui ne sont pas optimales. De plus, comme l'algorithme échantillonne à partir des commandes précédemment trouvées, une faible variance rend l'exploration de nouvelles situations difficile. D'un autre côté, l'utilisation de très grandes variances échantillonne tellement l'espace d'actions que les résultats précédants ne sont plus pris en compte et donc une grande quantité de trajectoires testées sont inutiles. En effet, une majorité des trajectoires se retrouvent en dehors de la piste. La figure 3.10 montre que l'augmentation de la variance améliore les performances en terme de coût de position jusqu'à ce qu'une variance trop élevée rende les résultats très inconsistants (au-delà de 1,5). Pour ce graphique le nombre d'échantillons est fixé à 1500 et l'horizon des trajectoires à 25 étapes.

L'évolution des vitesses moyennes montrée dans la figure 3.11 montre que la vitesse augmente avec la variance de l'échantillonnage. La trajectoire initiale autour de laquelle les commandes sont échantillonnées est une trajectoire de commandes nulles. Il

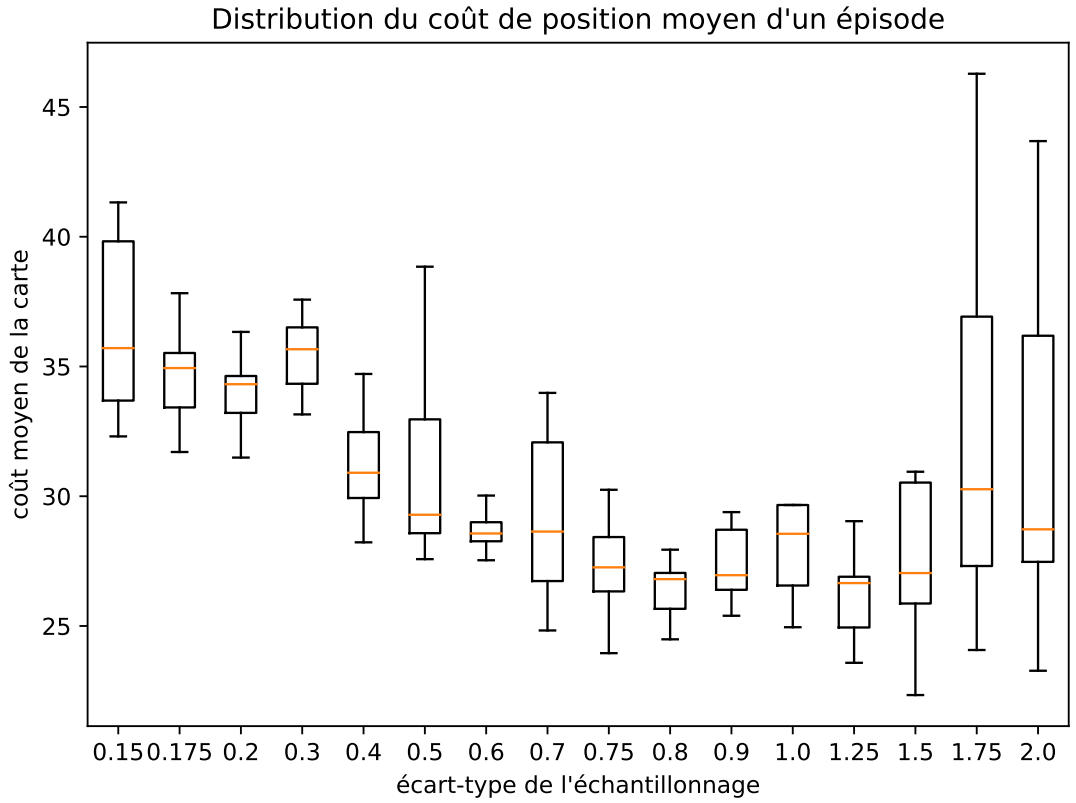


FIGURE 3.10 – Résultats moyens du suivi de piste en fonction de l'écart-type d'échantillonnage pour un bateau en simulation.

est donc difficile de progresser vers des vitesses élevées à partir de celles-ci lorsque la variance est faible. En effet, comme les commandes ne sont pas beaucoup modifiées, elles restent proches de zéro. Les variances entre 0,4 et 1,25 semblent donner des résultats satisfaisants. Au-delà de ces valeurs, les résultats de suivi de la piste sont très irréguliers et les vitesses apparemment satisfaisantes sont en fait le résultat du bateau occilant autour de la trajectoire.

3.5 Conclusion

La commande prédictive étudiée dans ce chapitre montre des résultats prometteurs pour le contrôle de robots mobiles pour des tâches complexes. Les principaux paramètres de l'algorithme ont été discutés. Cet algorithme de contrôle nécessite un modèle du système. La qualité de ce modèle est crucial pour la réussite du contrôle. Le chapitre suivant s'intéresse au problème de la modélisation de systèmes complexes.

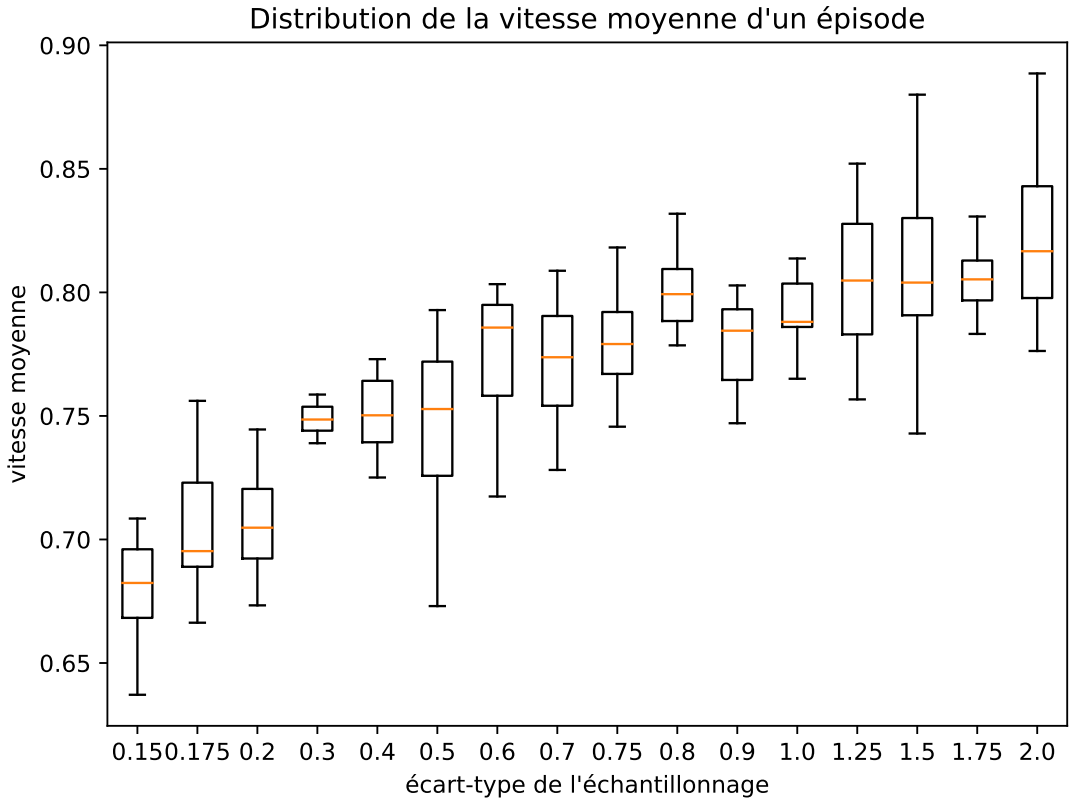


FIGURE 3.11 – Vitesses moyennes en fonction de l'écart-type d'échantillonnage pour un bateau en simulation.

Chapitre 4

Identification de modèle et apprentissage automatique

Pour prédire l'évolution d'un système dont le modèle physique est inconnu ou lorsque ce dernier est trop complexe pour être calculé en un temps raisonnable, des méthodes d'identification en boîtes noires sont nécessaires. Dans ce cas, la construction d'un modèle du système se fait uniquement à l'aide de l'observation des données entrantes et sortantes du système. Ces méthodes d'identification ont été largement étudiées et mises en application, notamment pour des systèmes linéaires. Cependant lorsqu'il est question de systèmes non-linéaires, ces méthodes deviennent rapidement complexes à mettre en place. L'objet de ce chapitre est d'étudier la pertinence de l'utilisation des méthodes d'apprentissage automatique comme outil d'identification de système.

4.1 Identification de système

L'objectif de l'identification est de construire un modèle du système qui permette de déduire l'évolution de ce dernier. Étant donné l'historique de l'évolution du système et de la commande à laquelle il est soumis, la question est de connaître son état futur. Pour construire ce modèle, seule l'observation du système en fonctionnement doit intervenir. Le modèle doit être le résultat de l'étude des données collectées lors de son fonctionnement.

4.1.1 Description de l'identification de système

L'objectif de l'identification de système est de construire un modèle qui permette de prédire son évolution.

Soit l'état du système $X(t)$ regroupant l'ensemble des variables permettant de décrire le système. L'objet de l'étude est l'évolution du système au cours du temps, plus précisément la prédiction de l'état du système futur connaissant les états du système passés et les commandes appliquées au système.

Soit F le modèle recherché, l'équation (4.1) le décrit formellement pour un historique de n pas.

$$\begin{cases} \{X(t), X(t-dt), ..., X(t-n*dt)\} \\ \{U(t), U(t-dt), ..., U(t-n*dt)\} \end{cases} \xrightarrow{F} X(t+dt) \tag{4.1}$$

Le principe de l'identification est de déduire le modèle recherché de l'observation du système en fonctionnement. Il s'agit donc dans un premier temps de collecter les données du robot en fonctionnement puis d'utiliser une méthode de régression pour calculer le modèle.

Ce type de modèle pourra ensuite être utilisé dans des contrôleurs comme la commande prédictive décrite au chapitre précédent.

4.1.2 Solutions pour les systèmes linéaires

Les méthodes de type ARMA sont utilisées depuis des décennies avec succès. Considérant la commande $U(t)$ et la sortie $X(t)$, le modèle ARMA du système est donné par l'équation aux différences finies suivante :

$$X(t) + \sum_{k=1}^{p} a_k X(t-k) = \sum_{k=1}^{q} b_k U(t-k). \tag{4.2}$$

Généralement l'objectif est de déterminer la valeur de la sortie du système au prochain pas de temps étant donné la sortie précédente, l'entrée et les paramètres $\{a_1, ..., a_p, b_1, ...b_q\}$:

$$X(t) = -\sum_{k=1}^{p} a_k X(t-k) + \sum_{k=1}^{q} b_k U(t-k). \tag{4.3}$$

La linéarité du modèle permet de facilement calculer les paramètres optimaux en utilisant une régression des moindres carrés sur une série d'observations faites du système en fonctionnement.

4.1.3 Difficultés des approches généralistes

Malheureusement tous les systèmes ne sont pas linéaires. Les frictions et les chocs sont des exemples courants de non-linéarité. Or l'extension des modèles régressifs à des systèmes non linéaires est complexe. Il est nécessaire de faire appel à des méthodes différentes, méthodes qui doivent se spécialiser sur les systèmes étudiés. Pour régler les paramètres de ces méthodes spécialisées, il faut faire appel à une expertise des systèmes en question. Il serait donc souhaitable d'avoir une méthodologie applicable aux différents types de robot de façon équivalente. C'est l'intérêt des méthodes d'apprentissages automatiques.

4.2 Apprentissage automatique

L'objet de l'apprentissage automatique est l'extraction de connaissance de corpus de données. Il s'agit d'extraire des informations de corpus connus afin d'inférer des connaissances utilisables sur de nouvelles données (de nouvelles observations). Par exemple, on souhaite extraire d'un ensemble de données de navigation la dynamique d'un drone afin de pouvoir prédire sa trajectoire de l'observation des données de positions et de vitesses présentes. Le but de l'apprentissage est la compréhension et la prédiction (Friedman, Hastie et Tibshirani, 2001). L'apprentissage peut être divisé en plusieurs catégories dont l'apprentissage supervisé, non supervisé, en ligne ou par renforcement. Ici la catégorie qui nous intéresse est celle de l'apprentissage supervisé, en particulier le problème de la régression.

4.2.1 Description du principe général

Les réseaux de neurones sont un outil permettant de représenter une large gamme de fonctions, en particulier les fonctions non-linéaires qui sont traditionnellement difficiles à traiter. Récemment, l'utilisation de réseaux de neurones complexes couplée

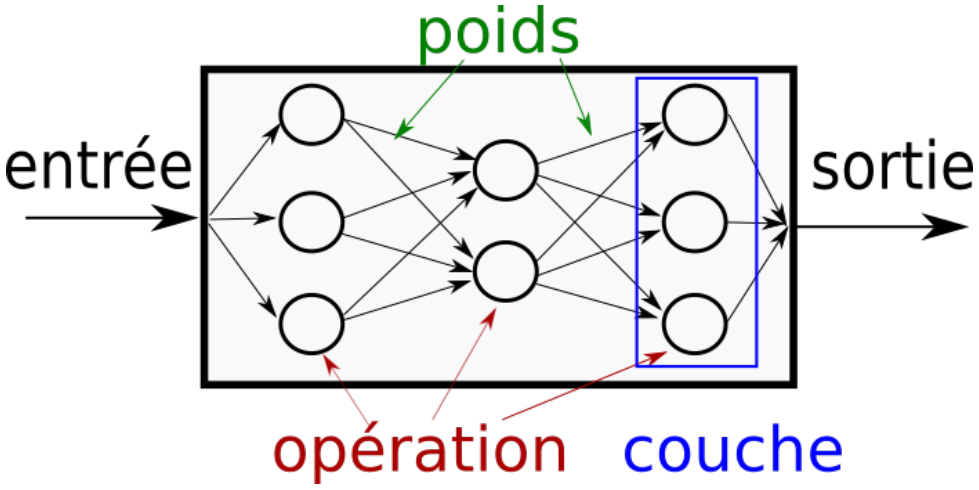


FIGURE 4.1 – Schéma d'un réseau de neurones.

à l'amélioration des performances des algorithmes et du matériel informatique a permis de produire d'impressionnants résultats (GOODFELLOW, BENGIO et COURVILLE, 2016).

L'identification de système peut être vue comme une application de l'apprentissage automatique au problème particulier de la prédiction de l'évolution de système dynamique. Il est ainsi naturel de s'inspirer des progrès de l'apprentissage automatique, en particulier l'utilisation de réseaux de neurones, pour réaliser l'identification de systèmes complexes.

Les réseaux de neurones sont constitués de nœuds, appelés neurones, effectuant des opérations. Ces nœuds sont reliés entre eux via des connections dont l'importance varie. Afin de simplifier la représentation de ces réseaux, ces nœuds ou neurones sont regroupés par couches. Une couche est un ensemble de neurones performant la même opération en même temps. En figure 4.1, un schéma représente les différents éléments constituant le réseau de neurones.

Le choix des opérations, du nombre de couches et de la taille des couches définit une architecture. Le choix de cette architecture relève de l'expert. Ce choix détermine en grande partie la capacité de représentation du réseau. Les poids des liaisons, quant à eux, sont des paramètres. L'entraînement consiste à faire varier ces paramètres afin de minimiser l'erreur sur le corpus d'entraînement. Le nombre de paramètres (connexions) donne une idée de la complexité du réseau. Plus un modèle comprend de paramètres, plus il est complexe.

4.2.2 Fonctionnement de l'entraînement

Étant donné un corpus composé de paires d'entrées-sorties connues, une fonction objectif (*loss function* en anglais) proportionnelle aux erreurs effectuées par le réseau de neurones est définie. Cela permet d'évaluer la distance entre les résultats obtenus par le modèle et la distribution des résultats souhaités (ceux présents dans le corpus). L'entraînement consiste à optimiser cette fonction objectif sur l'ensemble des paramètres possibles. Pour cette recherche de minimum, résolue par descente de gradient, la méthode ADAM, développée par KINGMA et BA, 2014 est utilisée ici.

Lors de l'utilisation de corpus, il est important de faire la distinction entre les données utilisées pour l'entraînement (corpus d'entraînement) et les données consacrées

à l'évaluation des performances du réseau entraîné (corpus de test). De plus, lorsque les paramètres extérieurs aux réseaux de neurones dits hyperparamètres varient, il est nécessaire de considérer un corpus de validation. En effet, les performances du réseau pour chaque paramètre seront évaluées grâce au corpus de validation. Une fois les hyperparamètres choisis, les performances finales du réseau sont évaluées sur le corpus de test, données qui n'auront jamais été utilisées dans les phases d'entraînement. C'est le principe de la validation croisée. Cette distinction est extrêmement importante afin de minimiser les problèmes de surapprentissage. En effet, de bons résultats lorsque notre modèle n'est capable de réussir que dans les cas déjà vus n'est pas souhaitable. L'entraînement n'a d'intérêt que si le modèle est capable de généraliser.

4.2.3 Discussion à propos de l'architecture

De nombreuses architectures ont déjà été étudiées dans d'autres domaines d'application. Il est donc pertinent de réfléchir à l'architecture que l'on veut utiliser pour la tâche de modélisation. Dans ce domaine, les architectures les plus souvent rencontrées sont les perceptrons multi-couches (MLP). Cependant, de nombreuses autres approches sont utilisées pour la prédiction de séries temporelles. En particulier les réseaux convolutionnels à une dimension, *Convolutional Neural Networks* (CNN) (LeCun, Bengio et al., 1995), ainsi qu'une grande variété de réseaux récurrents, *Recurrent Neural Networks* (RNN) (Lipton, Berkowitz et Elkan, 2015), dont les réseaux récurrents à mémoire court et long termes, LSTM (Hochreiter et Schmidhuber, 1997) font partie. Plus récemment les réseaux d'attention semblent intéressants à étudier dans le cadre de la modélisation.

Les MLP sont une des architectures les plus communément utilisées pour l'identification de système. Ils sont basés sur l'empillement de couches denses. Ces réseaux relativement simples ont l'avantage d'avoir un temps d'inférence faible. Dans le cas de l'identification de système, les historiques de commande de vitesses sont des données temporelles. Cependant ces réseaux ne sont pas prévus pour gérer de telles structures de données. Cela présente des inconvénients. Premièrement, la longueur de l'historique considéré est un paramètre à décider lors de la conception du réseau. Deuxièmement, l'aspect séquentiel de ces données n'est pas pris en compte.

Les réseaux convolutionnels à une dimension résolvent partiellement ce problème. Ils sont constitués d'empillements de couches convolutionnelles. Par construction, ces réseaux traitent les séquences d'entrée de façon ordonnée. Ce traitement temporel est séduisant car il ressemble aux comportements des réseaux récurrents tout en étant plus simple. Le principal inconvénient de ces réseaux est que leur conception requiert plus d'hyperparamètres et qu'ils ont un temps d'inférence plus lent comparé au MLP.

Les réseaux récurrents sont un groupe de réseaux qui prend en compte explicitement l'aspect temporel de leur entrée. Ils utilisent une mémoire interne, aussi appelée état caché, qui leur permet de stocker le contexte courant. Ce type de stratégie a permis à ces réseaux d'obtenir de très bons résultats, notamment pour le traitement automatique des langues. Le principal inconvénient d'utiliser ces architectures plus spécialisées est la difficulté de gérer l'état caché afin de maximiser les performances du réseau.

Les avancées récentes dans le domaine du traitement automatique des langues ont mis en avant un nouveau type d'architecture : les modèles basés sur l'attention. Ce

mécanisme est à la base des architectures "Transformer" (Vaswani et al., 2017) et "Bert" (Devlin et al., 2018) récement développées. L'idée utilisée par cette méthode est d'apprendre quelles parties de la séquence d'entrée sont les plus influentes et donc où concentrer "l'attention" du réseau. Il semble justifié de tester des réseaux de neurones basés sur ce concept. En effet, le problème d'identification qui nécessite d'apprendre quelles parties de l'historique sont pertinentes pour l'inférence, semble pouvoir bénéficier des avantages de l'attention.

Cependant le mécanisme d'attention est très complexe et demande beaucoup de puissance de calcul. Il est donc difficile de l'utiliser sur des systèmes embarqués, particulièrement lorsque l'inférence doit être faite avec des contraintes de temps importantes comme c'est le cas dans notre algorithme de contrôle. Afin de répondre plus précisément à la problèmatique d'identification, Antoine Richard et moi avons mis au point une version allégée et plus spécialisée de cette architecture. La figure 4.2 montre l'architecture en question. Ce modèle traite quatre entrées : une séquence d'historique d'états, une séqence d'historique de commandes, un masque causal et une séquence de positions. Il produit en sortie la séquence des états futurs. Le masque causal est nécessaire pour garantir que le réseau ne corrèle pas l'élément courant avec des éléments futurs de la séquence pour faire ses prédictions. En effet en traitement du langage il est pertinent de rechercher des éléments dans la suite de la phrase lorsque l'on cherche à déterminer le sens d'un mot. Dans le cas du traitement de séries temporelles, lorsque l'objectif est de prédire les éléments suivants à partir des précédents, il est nécessaire de garantir la causalité du modèle. Pour prédire l'élément n, seules les informations concernant des éléments précédents sont utilisées. La séquence des positions est nécessaire pour transmettre à travers le réseau l'information de position de chaque élément à l'intérieur de la séquence traitée.

Pour prédire la sortie, le réseau exécute les opérations détaillées comme suit. Premièrement les séquences d'états et de commandes en entrée du modèle sont projetées dans un espace de plus grande dimention en utilisant une transformation linéaire. Cela permet de séparer les différentes variables plus facilement. L'encodage positionnel effectué ensuite permet d'intégrer l'information de position. Les séquences nouvellement encodées sont traitées séparement par une couche d'attention. Les sorties de chacune de ces têtes d'attention sont concaténées et traitées par la couche de propagation (*feed-forward*) constituée d'une pile de couches denses. Enfin le résultat est reprojeté dans la dimention de notre état en utilisant une simple couche dense sans fonction d'activation. Il a été choisi de séparer les séquences d'états et de commandes et de les traiter à l'aide de deux têtes d'attention différentes. Cela permet au réseau de naturellement et simplement considérer des horizons différents pour les deux séquences. La solution plus générale proposée dans l'architecture Transformer qui permet d'utiliser plusieurs têtes sur une même séquence et d'apprendre quelles parties doivent être traitées différemment est beaucoup plus complexe et coûteuse en calcul. L'utilisation de la connaissance du problème permet de grandement simplifier le réseau en attribuant manuellement une tête différente à la séquences des commandes et à celle des états.

4.2.4 Limites

La modélisation de systèmes via l'apprentissage automatique rencontre plusieurs écueuils. D'une part, ces méthodes sont coûteuses en temps de calcul. D'autre part, elles sont très gourmandes en données. Enfin elles sont dépendantes de la qualité et de la variété du corpus. À propos du temps de calcul, deux éléments viennent tempérer ce

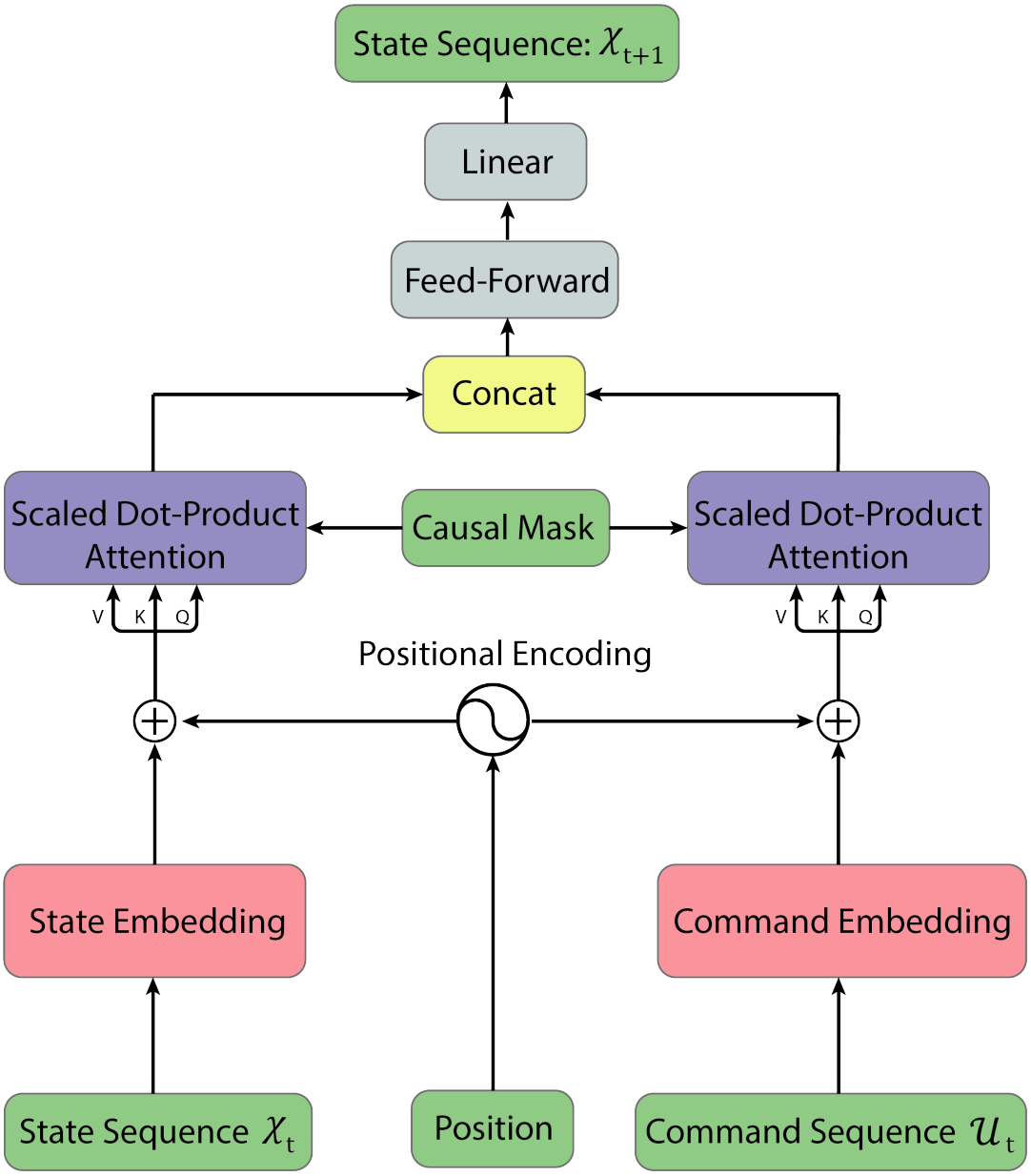


FIGURE 4.2 – Structure d'attention développée par Antoine Richard pour l'identification de système.

problème. D'abord, l'étape particulièrement coûteuse en temps de calcul est l'étape d'entraînement du réseau de neurones. Or ce travail ne nécessite pas d'être fait en temps réel. Les contraintes de temps ne sont donc pas strictes et il est possible de faire appel à l'utilisation de serveurs de calculs puissants. Ensuite, les avancées techniques permettent d'embarquer toujours plus de puissance de calcul à coût raisonnable.

La difficulté liée à la dépendance de cette méthode à de grandes quantités de données est un problème à part entière. Dans la suite, les difficultés rencontrées à ce sujet sont détaillées. Dans un premier temps, le problème est traité par la mise au point d'un protocole de génération de données permettant de générer des données suffisantes pour les cas d'utilisation. Dans un second temps, dans le chapitre 5, une méthode d'entraînement permettant de maximiser l'utilisation des données est exposée.

4.3 Collecte de données en robotique mobile

La nécessité de quantité de données importantes pour l'entraînement des réseaux pose la question de la génération de ces données. En effet, il est question ici de la modélisation de système robotique. Les données qui nous intéressent concerne donc l'évolution du système en fonctionnement.

4.3.1 Contraintes et coût de la collecte de données

Pour collecter des données d'un système en fonctionnement, il est nécessaire d'avoir au préalable une stratégie permettant de générer les situations d'exemples. Il peut s'agir d'un contrôleur connu mais limité qui permet de générer des données en vue de concevoir un contrôleur plus performant. Le problème est alors que la démonstration se limite aux situations connues et il va donc être difficile d'étendre le modèle appris aux zones où le contrôleur existant est inutilisable. Pour s'affranchir d'un contrôleur préexistant, un expert "humain" le remplace. La démonstration de l'expert permet de générer les données nécessaires à la modélisation. Cela présente deux problèmes. Premièrement, la quantité de données générées est contrainte par la disponibilité du démonstrateur. Deuxièmement l'expert aura tendance à faire cette démonstration dans les conditions qu'il connaît le mieux et non dans les cas difficiles. Afin de pallier au problème des zones inexplorées, il est possible d'imaginer des programmes sollicitant de façon aléatoire ou systématique le robot de sorte d'observer un maximum de situations. Outre le problème de l'étendue de la génération de données à faire dans ce cas, le problème de la sécurité des personnes et du matériel, est alors prépondérant.

4.3.2 À propos de la simulation

Pour modérer ce problème de collecte de données, le recours à la simulation est fréquent. D'une part le problème de sécurité n'est plus bloquant mais de surcroît la quantité de données générées est uniquement contrainte par la puissance de calcul à disposition. Malheureusement, il n'est pas facile de migrer de la simulation à la réalité. En effet le problème de transfert, utilisation d'un modèle appris en simulation dans des situations non simulées, est un champ de recherche à part entière Andrychowicz et al., 2020. Cependant, la simulation est une étape qui permet de valider efficacement les algorithmes employés.

Les logiciels de simulation en robotique comme gazebo (Koenig et Howard, 2004) ou V-REP (Rohmer, Singh et Freese, 2013) permettent de tester de façon pertinente un grand nombre de situations et de problèmes. Dans ces conditions, les limitations en nombre de données deviennent des limitations en temps de calcul, augmentant considérablement le champ de l'étude. Cependant il est nécessaire d'avoir à l'esprit les simplifications parfois importantes que l'environnement de simulation produit. Cela permet certes de tester de façon plus rapide et plus facile les solutions développées, mais ne remplace pas l'épreuve de la complexité d'une expérience sur le terrain. La simulation est un outil puissant pour préparer le terrain ou pour tester des nouvelles solutions mais ne permet pas d'attester le fonctionnement effectif du système en conditions réelles.

4.3.3 Déséquilibre des jeux de données

Comme vu précédemment au point 4.3.1, la collecte de données est un point difficile. De plus, un autre problème qui apparaît lors de la construction de corpus de données est leur déséquilibre. En effet la quantité de données seule ne suffit pas pour construire un corpus de bonne qualité. La variabilité et la richesse des données est un critère important. Il est difficile d'apprendre à un drone à tourner si le corpus considéré ne comporte que des exemples de drones allant en ligne droite. En robotique mobile, ce problème de déséquilibre de corpus apparaît rapidement. Comme discuté précédemment, la démonstration nécessaire doit être faite soit par un programme de contrôle déjà connu, soit par un expert. Or, l'algorithme de contrôle déjà connu sera à même de récolter des données dans les contextes maitrisés mais ne permettra pas d'avoir des informations sur les circonstances non prévues, circonstances qui nous intéressent particulièrement. Beaucoup de données sont redondantes dans les circonstances souvent rencontrées comme les situations de départ. Manquent alors les données dans des situations exceptionnelles. De plus, ces données exceptionnelles sont rarement obtenues dans les mêmes circonstances. D'autre part, le même problème surgit dans le cadre d'un utilisateur humain. Par exemple, si l'utilisateur humain manipule un drone pour tourner à grande vitesse, il va consacrer beaucoup de temps dans les conditions où il se sent à même de faire l'exercice souhaité. La quantité de données va rester faible dans les situations moins connues. Quels que soient les mouvements que le drone va faire, il va répéter plus de fois certains mouvements de base, dont les données sont déjà enregistrées.

Pour illustrer le propos, la figure 4.3 p. 41 montre la répartition des commandes obtenues pour un corpus typique récolté lors des expériences. Dans le cas considéré, le drone est commandé en vitesse. Le graphique représente un historique des données récoltées en fonction des amplitudes des vitesses commandées suivant les axes x et y. La majorité des données sont collectées pour des commandes nulles. De plus l'extrême majorité des données sont regroupées autour des axes $v_x = 0$ et $v_y = 0$.

4.3.4 Conclusion sur la collecte de données

La constitution et l'exploitation de corpus de données sont à la fois coûteuses et compliquées. Le problème de déséquilibre des corpus est un problème majeur. Des propositions de solutions seront étudiées dans le chapitre suivant.

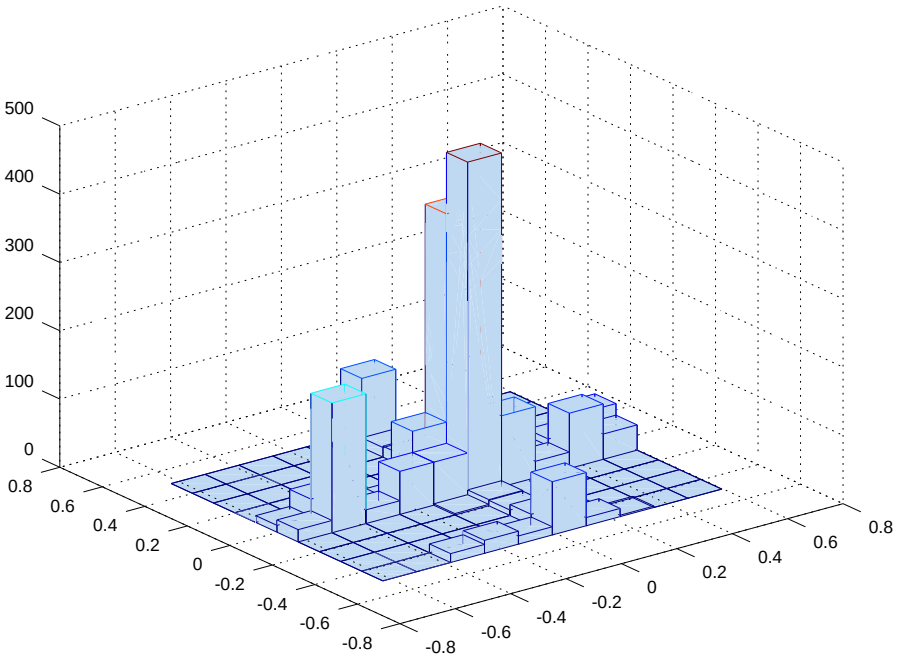


FIGURE 4.3 – Historique des commandes en fonction de leur composantes v_x et v_y.

4.4 Résultats

L'objet des expériences qui suivent est d'évaluer l'intérêt de l'utilisation de l'apprentissage automatique pour l'identification de système. Deux systèmes différents sont considérés : un drone et un bateau. Il s'agit de comparer les résultats de la modélisation de ces systèmes en utilisant d'une part les méthodes standards de l'identification ARMA et d'autre part une méthode d'entraînement de réseaux de neurones. Dans les deux cas, les résultats sont présentés en simulation puis sur le terrain. L'évaluations des méthodes se fait suivant le schéma : collecte de données, utilisation des différentes méthodes pour fabriquer un modèle puis évaluation des modèles obtenus.

4.4.1 Modélisation de la dynamique d'un drone

Le projet GRoNe vise à évaluer la pertinence du développement des drones dans la Grande Région. Dans ce cadre, il parait pertinent d'évaluer nos méthodes d'identification sur un modèle de drone couramment utilisé tel que le bebop 2.0 développé par Parrot [1].

En simulation

Génération des données

L'environnement de simulation choisi est le logiciel gazebo qui s'interface facilement avec le logiciel ROS pour la programmation des différents algorithmes de contrôle du drone. Pour la simulation du drone lui-même est utilisé le *tum simulator* [2] qui offre la même interface que le contrôleur bas niveau du drone de Parot. L'implémentation s'appuie largement sur ROS. Dans ces expériences, la version *kinetic* de ROS est utilisée. Le contrôleur fonctionne à une fréquence de 5 Hz, fréquence à laquelle le drone publie son odométrie. Les expériences sont faites en utilisant un ordinateur sous linux utilisant la distribution Ubuntu et ayant un CPU Intel i5 avec 8 GigaOctets de mémoire. Le système à identifier est l'ensemble composé du drone et du contrôleur bas niveau.

Dans un premier temps, un contrôleur basique est utilisé. Il s'agit d'un contrôleur Proportionel Integral Dérivé (PID) pour diriger le drone sur un carré défini par les positions de quatre points. Cela permet de générer facilement des données. Cependant cela ne permet pas de générer un corpus assez riche pour apprendre un modèle pertinent du drone.

Une deuxième méthode a été adoptée pour collecter des données. Pour générer un corpus qui explore mieux son espace d'action, un contrôleur simpliste est imaginé. Il s'agit d'un contrôleur qui va dans une direction aléatoire avec une vitesse aléatoire. Pour des raisons de stabilité et pour éviter que le drone ne s'écrase, des contraintes ont été mises en place. Ce programme est fait pour échantillonner de façon uniforme l'espace des actions. Du fait des contraintes de sécurité, la dynamique verticale est légèrement plus représentée. En effet, pour éviter qu'il ne s'écrase, le drone est forcé de regagner de l'altitude lorsque celle-ci est trop basse.

1. Copyright ©2016 Parrot Drones SA. All Rights Reserved
2. http://http://wiki.ros.2org/tum_simulator

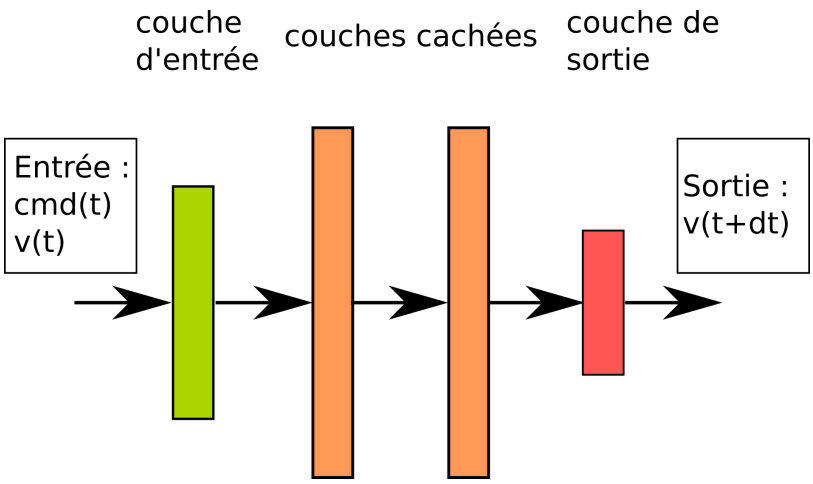


FIGURE 4.4 – Perceptron multi-couches.

Observations des résultats sur différentes dynamiques

Modéliser la dynamique du système est l'objectif. Les entrées utilisées sont donc les vitesses (v_x, v_y, v_z, v_{rz}) et les commandes (u_x, u_y, u_z, u_{rz}). L'historique utilisé est de deux pas. Par exemple les deux instants $(t, t-dt)$ pour prédire l'étape $(t+dt)$. Dans ces conditions, la période est dt. La sortie du modèle F est la vitesse à l'instant suivant. Le modèle est décrit dans l'équation 4.4.

$$\begin{cases} v_x, v_y, v_z, v_{rz}(t-dt) \\ u_x, u_y, u_z, u_{rz}(t-dt) \\ v_x, v_y, v_z, v_{rz}(t) \\ u_x, u_y, u_z, u_{rz}(t) \end{cases} \xrightarrow{F} v_x, v_y, v_z, v_{rz}(t+dt) \tag{4.4}$$

Pour résoudre ce problème, deux méthodes de modélisation sont utilisées. Le modèle qui sert de référence est un modèle ARMA du second ordre.

Le réseau de neurones utilisé est composé de la succession d'une couche d'entrées, de deux couches cachées denses et d'une couche de sorties, comme décrit en figure 4.4.

La couche d'entrées est composée de 16 nœuds qui correspondent aux variables d'entrée décrites dans l'équation précédente. Les couches cachées ont 32 nœuds chacune et la couche de sorties en a 4 qui correspondent aux variables de sortie décrites précédemment. Les deux couches cachées utilisent des activations *Rectified Linear Unit* (ReLU). La couche de sorties utilise une activation linéaire. La fonction objectif (*loss*) utilisée pour l'entraînement est l'erreur moyenne au carré et l'optimiseur est ADAM KINGMA et BA, 2014. Pour finir, l'implémentation du réseau de neurones utilise l'outil Keras en python qui utilise lui-même la bibliothèque Tensorflow ABADI et al., 2016.

La figure 4.5 décrit l'évolution des vitesses suivant l'axe x. Les deux modèles sont très bons. Ces résultats sont attendus car la dynamique de cet axe est linéaire. L'erreur moyenne sur le corpus de tests pour le réseau de neurones est de 0.064 m/s selon l'axe x. C'est deux fois meilleur que les performances sur le modèle ARMA qui, dans les mêmes circonstances, obtient une erreur de 0.146 m/s suivant le même axe. C'est une erreur tout à fait raisonnable. Dans ces conditions cela se traduirait par une erreur de position de 3 cm par pas de temps.

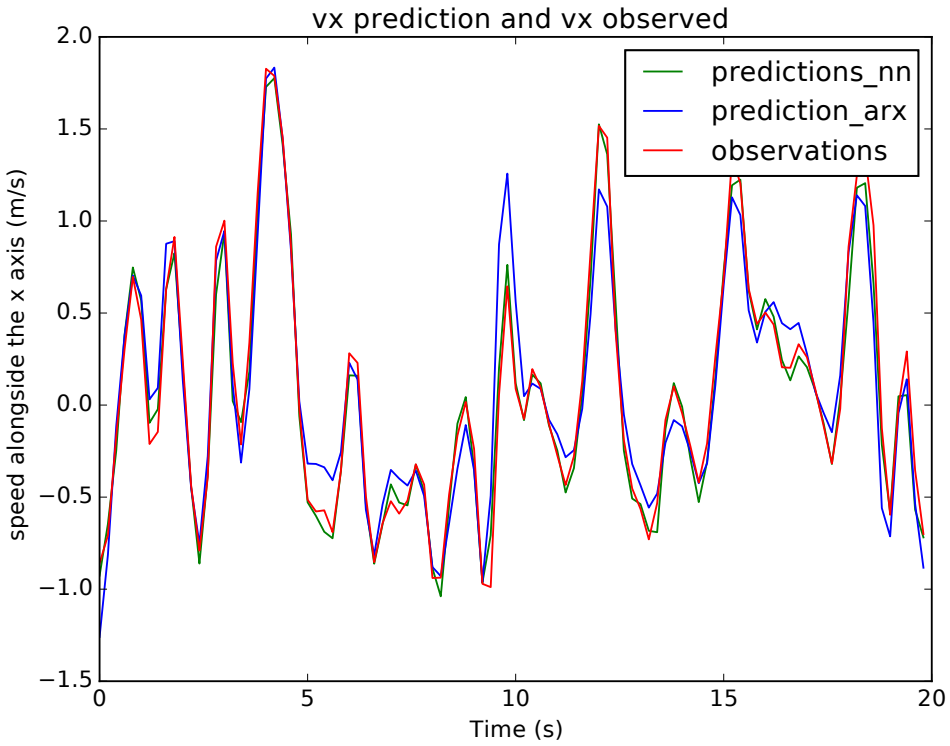


FIGURE 4.5 – comparaison des modèles autorégressif (ARX) et neuronal (NN) sur une dynamique linéaire.

La figure 4.6 s'intéresse à une dynamique plus complexe, la dynamique de la vitesse verticale. En effet, dans ce cas, le contrôleur bas niveau doit contrebalancer l'effet de la gravité. Dans ce contexte, la différence de performance entre le réseau de neurones et le modèle autorégressif est beaucoup plus importante. Le réseau de neurones est trois fois meilleur en terme quantitatif. L'erreur moyenne pour le réseau de neurones sur la vitesse verticale est de 0.045 m/s. Pour le modèle ARMA, elle est de 0.107 m/s. L'amplitude des vitesses verticales (de -0.6 m/s à 0.6 m/s) est plus étroite que celle pour la vitesse linéaire (de -1.5 m/s à 2 m/s). Une fois de plus, ce résultat est attendu puisque la méthode autorégressive utilisée est conçue pour des systèmes linéaires. L'intérêt est de démontrer ici qu'avec une même méthode, à base de réseaux de neurones, il est possible de modéliser des dynamiques non-linéaires sans que cela n'affecte les performances du réseau dans des cas linéaires.

Résultats en validation croisée

Pour poursuivre l'évaluation de notre méthode, elle est évaluée sur un corpus plus conséquent. Pour l'évaluation deux métriques différentes sont utilisées. La première, prédiction à un pas, représente la précision de la prédiction du prochain point. Sa valeur est calculée en utilisant la *Mean Squared Error* (MSE) sur toutes les prédictions faites pour le corpus de tests. Cette méthode permet d'évaluer la précision de la prédiction immédiate. La deuxième métrique concerne la précision de la prédiction en fin de trajectoire. On l'appelle précision à plusieurs pas. Dans ce cas il est nécessaire de préciser un horizon de temps qui détermine la longueur de la trajectoire évaluée. Ici, la longueur de la trajectoire choisie est de 15 pas. Pour calculer cette trajectoire, les modèles doivent s'appuyer sur leurs propres prédictions en terme de vitesse. Dans

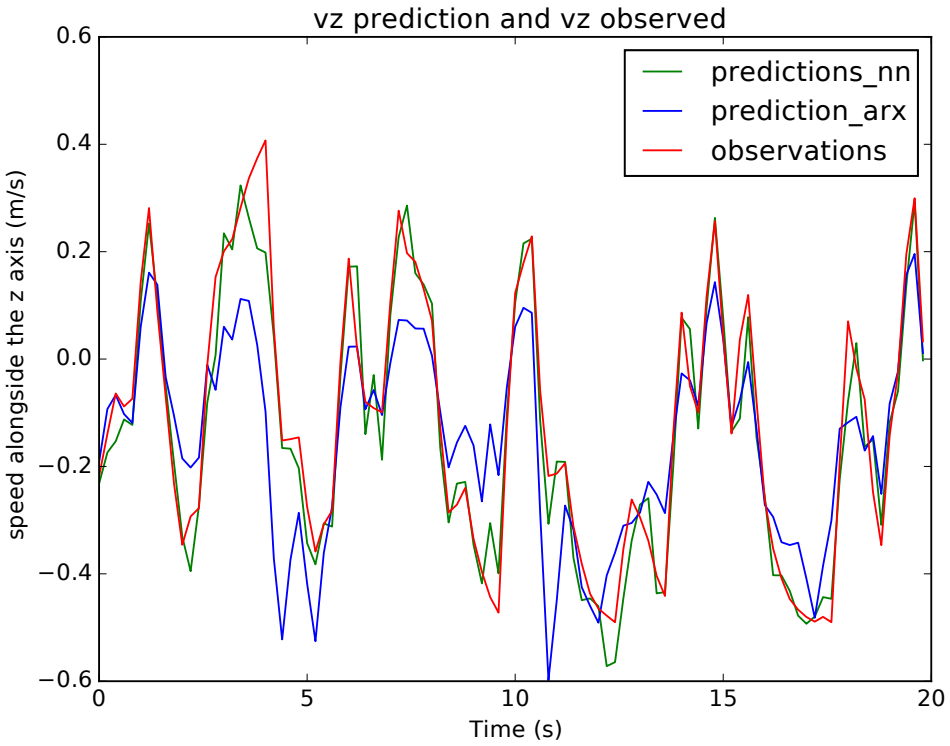


FIGURE 4.6 – comparaison des modèles autoregressif (ARX) et neuronal (NN) sur une dynamique non-linéaire.

le cas de l'évaluation sur une trajectoire, l'historique des commandes est fourni pour l'ensemble des 15 pas de la trajectoire à prédire. Cependant, seul l'historique des vitesses pour le premier pas est donné. Pour les 14 pas suivants, l'historique des vitesses est actualisé grâce aux prédictions faites par le modèle lui-même.

Afin de valider la méthode sur différents corpus de tests, les données à notre disposition sont divisées en cinq parties égales. L'entraînement est réalisé cinq fois en choisissant à chaque fois un nouveau corpus pour le test et en s'entraînant sur le reste. Chaque entraînement est une nouvelle expérience où l'entraînement repart de zéro. Il s'agit de s'assurer que les résultats ne dépendent pas d'un corpus de tests en particulier. D'autre part, les expériences sont toutes reproduites 10 fois. Dans le tableau 4.1 est présenté la moyenne des résultats obtenus : la méthode par apprentissage obtiens systématiquement de meilleurs résultats que la méthode classique. En erreur à un pas, les réseaux de neurones ont une erreur deux fois moindre même dans le cas le plus défavorable (cas du corpus de test numéro 4). Ces performances se reflètent dans l'erreur à plusieurs pas où l'écart est encore plus important. Dans ce cas, la méthode par apprentissage à une erreur de l'ordre de quatre fois plus faible que la méthode ARMA. Encore une fois, le cas du corpus de test 4 fait exception : le gain des réseaux de neurones n'est que de 20 %.

Sur le terrain

La méthodologie a été validée en simulation et reste maintenant à l'évaluer dans le cas réel, sur le terrain.

Test set	0	1	2	3	4
	Erreur à un pas				
ARMA	0.836	0.873	0.850	0.855	0.883
Réseau de neurones	0.311	0.346	0.109	0.329	0.481
	Erreur à plusieurs pas				
ARMA	1.003	0.986	1.056	0.983	0.958
Réseau de neurones	0.214	0.263	0.166	0.257	0.709

TABLE 4.1 – Résultat en validation croisée pour le drone simulé

FIGURE 4.7 – Le drone Parrot utilisé, équipé du gps RTK.

Le drone bebop de Parrot est utilisé pour la collecte des données sur le terrain. Afin de s'assurer une grande précision de nos mesures d'odométrie, le drone est équipé d'un gps RTK construit par emlid [3]. Le système de gps RTK utilisé un gps fixe et un gps mobile. Le gps fixe sert de référence et permet de corriger la dérive des données du gps mobile embarqué sur le robot. La figure 4.7 montre le drone utilisé équipé de son gps. Cet ajout a demandé quelques modifications du drone original. Pour des raisons de poids il n'était pas possible de rajouter une batterie pour le gps. Une dérivation de la batterie du drone vers le gps a donc été mise en place.

L'acquisition est faite en utilisant le mode RTK mais sans utiliser le *Post Processing Kinematic* (PPK). En effet ce mode n'est pas utilisable pour un algorithme en temps réel mais seulement de façon asynchrone. L'ensemble logiciel utilisé lors des expérimentations est comme suit : sur le drone, les contrôleurs de bas niveaux du constructeur (Parrot) fonctionnent sur l'ordinateur de bord du drone. Ils communiquent avec le package ROS bebop-autonomy [4] qui, lui, fonctionne sur l'ordinateur "base". Plusieurs logiciels différents interagissent entre eux sur l'ordinateur base. ROS est chef d'orchestre : il fait le lien entre tous les modules (packet bebob-autonomy, packet gps, collecte de données). Sur le drone, le contrôleur du constructeur communique, via le réseau wifi, avec le package bebop. Sur le gps, le logiciel du constructeur

3. https://emlid.com/reach/
4. https://bebop-autonomy.readthedocs.io/en/latest/

communique lui aussi avec l'ordinateur base via le réseau wifi. Sur l'ordinateur le package ROS dédié au gps se charge d'interpréter les données.

La modélisation est évaluée ici de la même façon qu'en simulation. Pour prédire une étape, les modèles s'appuient sur l'historique des vitesses et des commandes.

Une expérience consiste à l'entraînement d'un modèle sur le corpus d'entraînement puis à l'évaluation de ce modèle sur le corpus de tests. Tous les corpus n'étant pas égaux, il est possible qu'un corpus de tests facile donne de bons résultats. Afin de garantir que les résultats ne sont pas issus d'un tel hasard, plusieurs expériences sont effectuées en changeant le découpage entraînement-évaluation à l'intérieur du corpus considéré. En particulier dans les expériences suivantes, le choix a été fait de découper le corpus original en quatre parties. Tour à tour une partie est sélectionée comme ensemble de tests et les 3 parties restantes sont utilisées pour l'entraînement. Les méthodes à base de neurones sont parfois critiquées sur la variabilité de leurs résultats. Chaque expérience a donc été faite 10 fois de sorte de s'assurer que les résultats sont fiables. La moyenne de ces résultats est présentée.

Test set	0	1	2	3
	Erreur à un pas			
ARMA	0.778	0.812	0.783	0.804
Réseau de neurones	0.179	0.549	0.144	0.116

TABLE 4.2 – Résultat de l'évaluation à un pas en validation croisée pour le drone réel.

Les résultats présentés dans les tableaux sont des mesures d'erreur. Plus l'erreur est faible, meilleur est le résultat. D'un point de vue général : dans tous les cas, évaluation à un pas ou à plusieurs pas, la méthode basée sur l'entraînement de réseaux de neurones à de meilleures performances.

Ce résultat se généralise à chaque ensemble de tests, la méthode basée sur l'entraînement de réseau de neurones est supérieure à la méthode ARMA.

Les résultats obtenus pour l'erreur à un pas sont décrits dans le tableau 4.2 Dans le pire cas (cas du corpus de test 1), la solution basée sur les réseaux de neurones est 50% plus performante que la solution classique. Dans les autres cas, le gain obtenu par la méthode d'apprentissage est encore supérieur : le gain est de l'ordre de 400%. Les faibles résultats obtenus par ARMA peuvent s'expliquer par la difficulté de cette méthode à modéliser la dynamique verticale qui, comme discutée précédemment n'est pas tout à fait linéaire. La difficulté rencontrée par le réseau de neurones dans le cas de l'ensemble de test 1 peut s'expliquer par des conditions météorologiques (vent) qui rendent la dynamique du drone stochastique. Or, la méthode utilisée n'est pas prévue, pour gérer autre chose que des systèmes déterministes. Il serait donc nécessaire de considérer d'autres méthodes de modélisation pour traiter ce type de corpus.

Les bonnes performances des réseaux de neurones à un pas se retrouvent dans l'évaluation sur plusieurs pas. Le tableau 4.3 montre que dans chacun des cas la méthode basée sur l'apprentissage améliore les performances de 20 et 50 % par rapport à la méthode autorégressive. Dans l'ensemble, les erreurs mesurées sur des trajectoires sont plus élevées que dans le cas à un pas. Ce n'est pas surprenant puisque sur une

Test set	0	1	2	3
	Erreur à plusieurs pas			
ARMA	1.054	1.012	1.046	1.014
Réseau de neurones	0.863	0.738	0.543	0.804

TABLE 4.3 – Résultat pour l'évaluation sur trajectoire de la validation croisée pour le drone réel.

trajectoire, les erreurs de prédictions se cumulent au fur et à mesure que les prédictions sont faites.

Les méthodes d'apprentissages ont démontré leur capacité à modéliser un drone. Cependant, le système considéré inclut le contrôleur bas niveau du constructeur, ce qui simplifie grandement la dynamique du modèle à apprendre. Afin de tester notre méthode dans des situations plus adverses, il est intéressant de considérer des systèmes dont la dynamique est plus complexe : retard dans la dynamique, non-linéarité plus marquée. C'est l'objet des expérimentations suivantes qui sont faites avec un bateau.

4.4.2 Modélisation de la dynamique d'un bateau

Le drone utilisé précédemment ne permettait pas de s'interfacer directement avec les commandes moteurs du robot. Le choix a donc été fait de travailler sur une autre plateforme pour laquelle nous avions les capacités et les compétences techniques de commander directement les moteurs. Dans le cas du drone, le modèle prenait en entrée les vitesses et les commandes. Ces commandes étaient des commandes en vitesse destinées au contrôleur de bas niveau de Parrot©. Dans ce nouveau contexte, les commandes envoyées sont des commandes d'intensité directement à destination des contrôleurs des moteurs du bateau. De plus le bateau présente une dynamique plus complexe du fait de son inertie. La difficulté rencontrée pour tourner rend la planification d'autant plus cruciale.

En simulation

L'environnement de simulation utilisé pour le bateau est le même que pour le drone : gazebo est utilisé de pair avec ROS afin d'implémenter les expériences. Là où le packet bebop était utilisé pour le drone, le packet heron est utilisé pour le bateau, package développé par Clearpath Robotics. Le packet UUV simulator remplace le tum simulator.

Afin d'assembler des corpus de données comparables à ceux obtenus pour le drone, le contrôleur simpliste a été adapté pour qu'il envoie des commandes aléatoires aux moteurs du bateau. Afin de pouvoir capturer la dynamique plus lente du bateau, les commandes envoyées sont appliquées pendant un temps lui-même variable. Le corpus utilisé a été généré en laissant le contrôleur simple fonctionner pendant vingt heures. Afin de rester dans des conditions proches du réel, la fréquence de 5 Hz est conservée pour l'enregistrement de l'odométrie et le fonctionnement du contrôleur. En effet, dans le cas réel le gps fournit une odométrie à cette fréquence. Les entrées utilisées sont un historique des vitesses linéaire(v_{lin}), transversale (v_{tra}) et angulaire (ω_z) et un historique des commandes envoyées aux deux turbines du bateau (u_1, u_2). La sortie du modèle est composée des vitesses futures ($v_{lin}, v_{tra}, \omega_z$). L'historique choisi dans le cas des expériences est de douze pas, ce qui correspond à 2.4 s, afin d'être sûr de capturer

la dynamique du bateau. En effet dans le cas du drone il était possible se contenter de deux pas de temps d'historique soit 0.4 s, qui était largement supérieur à la réactivité du drone. En revanche la dynamique du bateau est beaucoup plus lente, de l'ordre de la seconde. Il est donc nécessaire d'utiliser un historique plus long. Un modèle basé sur un tel corpus devrait contenir des non-linéarités importantes car l'efficacité non-linéaire des turbines est prise en compte par le simulateur de même que la résistance de l'eau. Du fait de ces non-linéarités importantes du corpus, la méthode ARMA précédemment utilisée ne fonctionne plus. En effet, sur un tel corpus, la méthode régressive diverge. Ici l'objectif est de démontrer l'applicabilité de la méthode par apprentissage sur des systèmes complexes. Les résultats obtenus par une telle modélisation serviront de base de comparaison pour le chapitre suivant.

Les performances des modèles issus de l'entraînement de réseaux de neurones sont étudiés en termes de racine carré de l'erreur quadratique moyenne (RMSE) à un pas puis sur une trajectoire. Comme vu dans le paragraphe 4.2.3, plusieurs structures sont possibles pour l'architecture des modèles neuronaux. Les architectures les plus prometteuses ont donc été testées. Les résultats de ces tests en simulation sont présentés dans le tableau 4.4.

Architectures	RMSE à un pas moyenne	RMSE à plusieurs pas moyenne	Temps (s) forward
MLP	0.135	0.213	**0.0018**
CNN	0.133	0.214	0.0121
RNN	0.072	0.101	0.0103
LSTM	0.069	0.106	0.0210
GRU	0.072	0.108	0.0198
ATTN	**0.063**	**0.091**	0.0105

TABLE 4.4 – Performances des différentes architectures sur le corpus de simulation. RMSE : plus l'erreur est faible meilleur est le modèle. Temps : plus le temps d'inférence est faible plus le modèle est facile à utiliser.

Les méthodes basées sur l'entraînement de réseaux de neurones réussissent toutes à apprendre la dynamnique du bateau. En terme d'erreur moyenne à un pas, les modèles denses (MLP) et convolutionnels (CNN) obtiennent des résultats similaires. Les différents modèles récurrents (GRU, RNN, LSTM) ont tous des performances très semblables, de l'ordre de 50% supérieures aux modèles précédents. Le modèle basé sur le mécanisme de l'attention obtient les meilleures performances. Tous ces résultats se retrouvent sur l'erreur à plusieurs pas. Cependant on constate qu'en termes de temps de calcul, les modèles denses utilisés ici sont très supérieurs. Leur vitesse d'inférence est 5 à 10 fois plus rapide que les autres modèles.

Afin d'évaluer l'intérêt du modèle obtenu, on s'intéresse aux performances d'un contrôle à commandes prédictives utilisant ces modèles. Comme vu dans le chapitre précédent, les contraintes en temps de calcul pour le contrôle à commande prédictive à intégrale de parcours sont importantes. Une attention particulière a donc été portée à l'optimisation des calculs, notamment en faisant une utilisation avisée des bibliothèques numpy et TensorFlow (ABADI et al., 2016). De plus, pour une plus grande maîtrise sur l'implémentation des réseaux de neurones, la structure logicielle

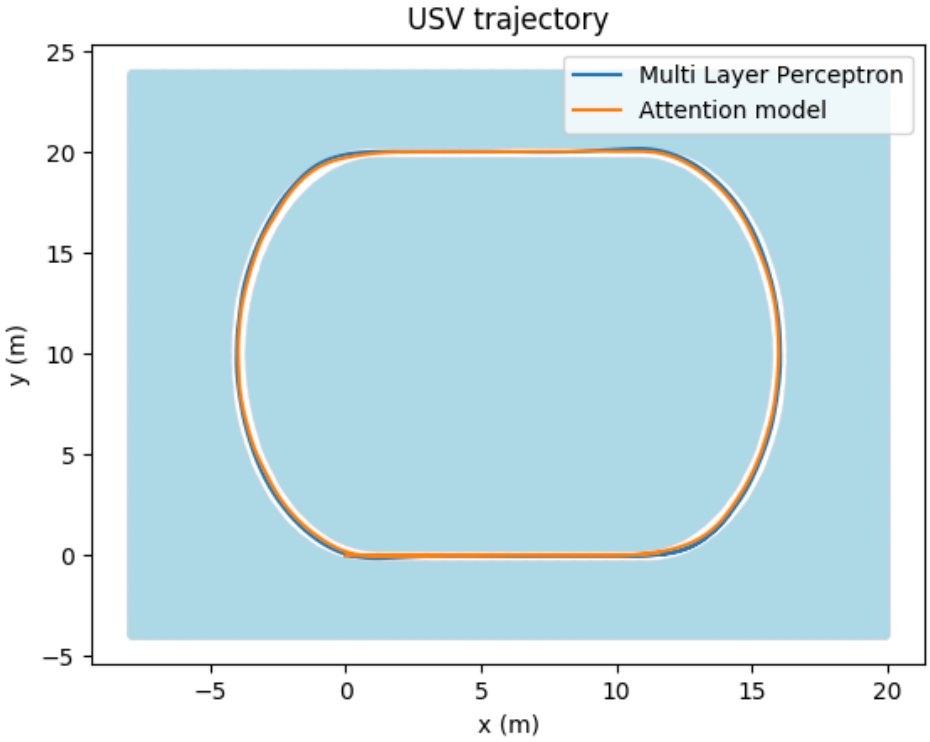


FIGURE 4.8 – Suivi d'une piste à l'aide de modèles appris.

keras (CHOLLET et al., 2015) n'a pas été utilisée ici, la programmation utilisant directement la bibliothèque TensorFlow a été préférée.

Pour évaluer la qualité des modèles dans le contexte du contrôle, les réseaux de neurones entraînés sont utilisés comme modèle dans un algorithme de commande prédictive. Le but est d'utiliser ce contrôleur pour compléter la tâche suivante : faire le tour d'une piste définie par une carte de coût. La figure 4.8 montre que l'algorithme de commande prédictive parvient à faire le tour d'une piste sans erreur malgré plusieurs changements de courbures. À ces changements de courbures, la trajectoire du bateau utilisant un MLP s'écarte du centre de la piste mais le contrôleur parvient à se rattraper sans sortir de la piste. Dans les mêmes conditions, entraînement avec les mêmes données et tâche à effectuer avec les mêmes paramètres, le modèle utilisant une architecture d'attention fonctionne encore mieux que le réseau simple. En particulier dans les zones difficiles, là où il y a changement de courbure, le réseau d'attention fait beaucoup moins d'écarts. Ce que l'on peut attribuer à une meilleure capacité de l'algorithme à anticiper grâce à un modèle plus précis. En effet tous les autres paramètres de l'algorithme sont équivalents.

Au cours des expérimentations, une erreur s'est glissée qui a débouché sur une observation intéressante. En voulant imposer une vitesse minimum au bateau, une erreur de signe a en fait imposé au bateau de se déplacer en marche arrière. La figure 4.9 montre que seul le réseau d'attention est assez bon pour réussir cette "nouvelle tâche" qui consiste à parcourir la piste en marche arrière.

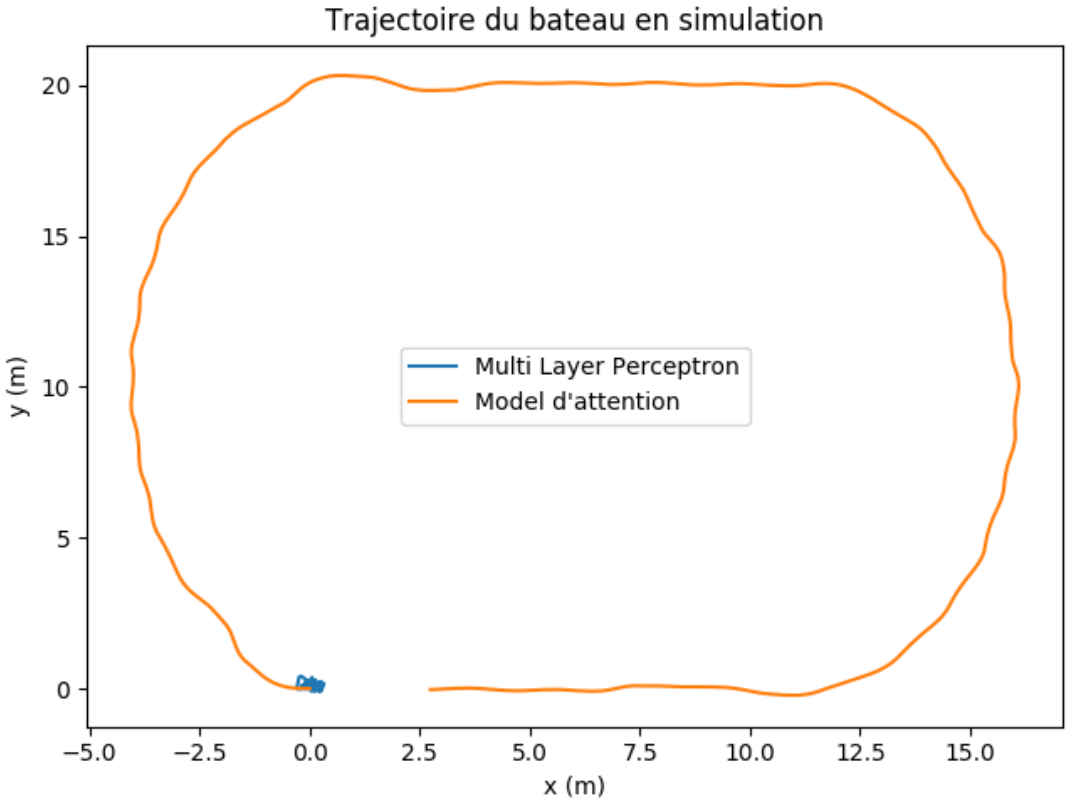


FIGURE 4.9 – Résultats du suivi de piste en marche arrière.

FIGURE 4.10 – Bateau Héron de clearpath utilisé dans les expériences

Sur le terrain

Poursuivant ces résultats prometteurs de simulation, la méthode est mise à l'épreuve du terrain. Pour cela, on utilise le bateau "héron" de Clearpath robotic. Contrairement au drone, l'intégralité des calculs doit être faite à bord. Il n'y a pas d'ordinateur de base. À bord du bateau, il y a deux ordinateurs. Le premier, équipé d'un processeur *atom* est chargé de l'orchestration ROS ainsi que de la communication avec les composants bas niveau (contrôleur moteur). Cet ordinateur est en communication via un câble *ethernet* avec le second ordinateur sur lequel fonctionnent les différents contrôleurs utilisés (contrôleur simplistique et contrôleur à commande prédictive). Ce second ordinateur est une carte de calcul *jetson* développée par Nvidia. De la même façon que pour le drone, afin d'obtenir une odométrie de grande précision, le bateau est équipé d'un gps RTK en communication directe avec la carte Nvidia. Pour collecter les données un équivalent du contrôleur simpliste de simulation est utilisé. Pour des raisons de sécurité, la collecte des données se fait sous la surveillance d'un utilisateur qui peut prendre la main sur les commandes du bateau à tout moment afin d'éviter tout accident.

Architectures	RMSE à un pas moyenne	RMSE à plusieurs pas moyenne	Time (s) forward
MLP	0.309	0.462	**0.0019**
CNN	0.310	0.537	0.0119
RNN	0.292	0.416	0.0103
LSTM	0.297	0.406	0.0210
GRU	0.291	0.432	0.0198
ATTN	**0.270**	**0.291**	0.0106

TABLE 4.5 – Performances des différentes architectures sur le corpus réel. RMSE : plus l'erreur est faible meilleur est le modèle. Temps : plus le temps d'inférence est faible plus le modèle est facile à utiliser.

Les performances des différents modèles sont détaillées dans le tableau 4.5. Les précisions des différents réseaux sont mesurées comme précédemment en reportant la racine carrée de l'erreur quadratique moyenne à un pas puis sur plusieurs pas. Les résultats sont moins marqués qu'en simulation. Cependant, les conclusions restent semblables. L'ensemble des modèles ont des performances satisfaisantes. Les modèles denses et convolutionnels ont des résultats très proches. Les modèles récurrents apportent ici une faible amélioration, de l'ordre de 10 %. Le modèle d'attention obtient les meilleurs résultats. En particulier, en terme d'erreur à plusieurs pas, le gain de ce dernier est particulièrement important, entre 30 et 40 %. Les conclusions en terme de calcul pour les différents modèles restent inchangées, les modèles MLP étant plus simples, ils restent plus rapides que les autres.

4.5 Conclusion sur l'apprentissage automatique

Dans ce chapitre, les capacités de modélisation des réseaux de neurones ont été montrées dans le cadre de l'identification de système. Leur utilité en simulation et sur des cas réels a été montrée pour deux systèmes très différents : un drone et un bateau. L'obtention de ces résultats a cependant mis en avant certaines difficultés. En particulier les impacts néfastes des déséquilibres dans les jeux de données ont été

discutés. Dans ce chapitre, le problème a été contourné en utilisant des corpus générés spécifiquement par des algorithmes conçus sur mesure. Cependant, l'exploration systématique de l'espace d'actions n'est pas toujours possible. Il est donc souhaitable de trouver une solution à l'exploitation de corpus déséquilibrés. Ce sera l'objectif du prochain chapitre.

Chapitre 5

Priorisation

Dans le chapitre précédent, l'intérêt de l'utilisation de réseaux de neurones pour la modélisation des systèmes a été établi. La quantité importante de données nécessaires pour ces méthodes d'apprentissage est le principal obstacle à leur mise en application, notamment en robotique. Ce chapitre porte sur l'étude de méthodes de traitement des données permettant d'alléger ce problème.

5.1 Motivation

L'utilisation des méthodes d'apprentissage basées sur des réseaux de neurones pour faire de la modélisation de système nécessite des corpus conséquents. Dans le chapitre précédent, il a été vu que la constitution de ces corpus de données est difficile. En particulier, ces corpus ont tendance à être déséquilibrés. En effet, ces corpus sont principalement constitués de données semblables car récoltées dans des situations similaires.

5.1.1 Conséquence du déséquilibre sur l'apprentissage

La quantité de données, nécessaire aux réseaux de neurones, est souvent mise en avant comme l'élément clé dans l'entraînement. Cependant, il est important de garder à l'esprit que la quantité de données seule ne suffit pas pour entraîner un modèle de qualité. Pour contrôler un système dans certaines situations, il est nécessaire d'avoir un modèle pertinent pour ces conditions. Or, cela n'est possible que si des données du système fonctionnant dans ces conditions sont présentes dans le corpus d'entraînement. Par exemple, dans le cadre de la modélisation de la dynamique d'un drone, il est vain d'essayer d'utiliser des données du drone se déplaçant dans le plan pour modéliser la dynamique verticale du drone.

Même lorsque le corpus contient des données variées, les techniques d'optimisation utilisées lors de l'entraînement peuvent porter préjudice à la qualité du modèle. Si certaines conditions sont beaucoup plus représentées que d'autres, le modèle résultant de l'entraînement sera extrêmement efficace dans ces conditions au détriment des autres situations. C'est le principal problème causé par le déséquilibrage observé dans les corpus.

Lors de l'utilisation d'un robot, certaines situations rencontrées fréquemment sont bien connues. D'autres, plus rares, présentent plus de difficultés. L'intérêt d'utiliser des méthodes d'apprentissage est souvent motivé par la volonté de modéliser ces cas moins connus. Ces cas plus complexes sont aussi, du fait de leur rareté, moins représentés dans les corpus. À l'inverse, les situations simples sont abondantes. Or, une amélioration sur les cas complexes implique un compromis sur les cas simples. Si les

cas simples constituent la majorité du corpus, comme c'est souvent le cas, ce compromis sera toujours en leur faveur. Un corpus contenant une majorité d'exemples pour un cas simple permet d'entraîner un modèle ayant une grande précision dans ces conditions. Ce modèle aura une précision globale pour l'entraînement même s'il répond au hasard sur les cas compliqués. Cependant lors de l'utilisation de ce modèle dans un contrôleur prédictif ces erreurs sur les cas compliqués peuvent s'avérer critiques. D'autre part, l'un des objectifs de ces modèles est de réussir à contrôler le système dans les cas complexes, ceux qui sont difficiles à enregistrer lors de la collecte des données. Il y a donc une différence entre la distribution des exemples de l'entraînement et la distribution cible pour l'utilisation du modèle.

5.1.2 Importance d'un exemple pour l'apprentissage

Au cours de l'apprentissage, lorsqu'un modèle rencontre une situation qu'il connaît bien, il n'évolue pas beaucoup. En effet la mise à jour de ses paramètres est proportionnelle à l'erreur de prédiction. Or, dans ce cas, l'erreur qu'il fait sur cet échantillon est très faible et il ne fait que valider l'état de ses paramètres. Dans le cas d'un échantillon correspondant à un cas nouveau, l'erreur du réseau est importante. Pour minimiser cette erreur, le modèle doit évoluer significativement. Il est clair qu'au cours de l'apprentissage certains exemples ont plus d'informations à apporter que d'autres au processus d'entraînement du modèle. Il serait souhaitable de concentrer l'apprentissage sur ces exemples riches en informations pour optimiser l'efficacité de l'entraînement. C'est l'objet des méthodes de priorisation qui essaient de sélectionner les données en ce sens.

Pour pouvoir prioriser certains échantillons lors de l'apprentissage, il est nécessaire d'avoir un critère de sélection. Une mesure de la pertinence d'un échantillon pour l'entraînement est donc nécessaire. Il y a plusieurs choix possibles pour cette mesure. Une première approche est de considérer que les exemples sur lesquels il y a le plus de choses à apprendre sont ceux pour lesquels les erreurs sont les plus grandes. C'est le choix qui est fait dans la méthode *Prioritized Experience Replay* (PER) (Schaul et al., 2015) détaillée plus loin.Une autre mesure de la qualité d'un exemple pour l'apprentissage peut être obtenue par l'observation de l'évolution des paramètres générés par l'observation de l'exemple. Ce choix de mesure est à la base de la méthode du gradient (Katharopoulos et Fleuret, 2017) détaillée plus loin.

5.2 La priorisation comme solution au déséquilibre des corpus

Les méthodes utilisées pour répondre au problème de déséquilibre des corpus décrit précédemment sont présentées dans cette section.

5.2.1 Rejoue priorisé des expériences

La problématique des corpus déséquilibrés a été rencontrée dans le contexte de l'apprentissage par renforcement notamment par l'algoritme DQN Mnih et al., 2013. Lors de sa recherche de solutions pour les jeux ATARI, cet algorithme s'entraîne en utilisant une mémoire tampon. Cependant les informations présentes dans cette mémoire sont réparties de façon déséquilibrée. En effet, les échantillons correspondent aux écrans successivement rencontrés dans une partie de jeu ATARI. Or la plupart des écrans se ressemblent. Seuls certains sont associés à des récompenses contenant

des informations pertinentes. De plus afin de maximiser l'utilisation des informations présentes dans la mémoire tampon, ces dernières sont rejouées plusieurs fois. Il a été montré par Schaul et al., 2015 que la priorisation de certains échantillons par rapport à d'autres permet d'accélérer l'entraînement en utilisant moins de données tout en conservant les performances. L'utilisation d'une méthode similaire pour la problématique de modélisation est proposée ici.

Pour prioriser les exemples, une mesure de leur importance est nécessaire. Évaluer la capacité d'apprentissage associée à un échantillon est une question très difficile. Dans le contexte de l'apprentissage par renforcement, l'erreur de différence temporelle est ce qui semble le choix le plus logique. Dans notre contexte, pour l'échantillon i, la distance (δ_i) entre la prédiction de notre modèle et l'observation est utilisée. Ce choix découle de l'idée que notre modèle a d'autant plus à apprendre d'un exemple qu'il échoue à le prédire. Ensuite notre corpus de départ est rééchantillonné selon la loi de probabilité $P(i)$ décrite par l'équation 5.1.

$$P(i) = \frac{\delta_i^\alpha}{\sum_k \delta_k^\alpha}. \tag{5.1}$$

Le nouveau corpus, qui est créé via l'opération de réénchantillonnage, est utilisé pour l'étape d'entraînement suivante. Le détail de ce procédé est donné dans l'algorithme 2.

Algorithme 2 Échantillonnage priorisé

Entrée: données,
  $DonnesDentrainement \leftarrow donnes$
  $PoidsDechantillonnage \leftarrow \emptyset$
  $F \leftarrow Train(trainingData)$
  N nombre d'échantillons dans *donnes*
  pour $i = 0$ to N **faire**
    $\delta_i \leftarrow \|Y_i - F(X_i, U_i)\|$
    $P(i) \leftarrow \frac{\delta_i^\alpha}{\sum_k \delta_k^\alpha}$
    $w_i \leftarrow \left(\frac{1}{N}\frac{1}{P(i)}\right)^\beta$
    $PoidsDechantillonnage \leftarrow \{w_i\}_{0 \leq i \leq N}$
    $DonnesDentrainement \leftarrow$ donnée échantillonées $d_i \sim P(i)$
  fin pour

Une des limitations de cette approche est qu'elle se concentre sur une petite partie des échantillons. Malgré le fait que cette concentration améliore l'efficacité de l'entraînement, cela engendre aussi des problèmes. Le fait de se concentrer sur un petit nombre de données augmente le risque de surapprentissage. D'autre part, dans le cas de corpus bruités, il est difficile de faire la différence entre les cas compliqués et les valeurs aberrantes. Les corpus considérés pour l'identification de système sont souvent bruités, puisqu'ils sont dépendants de la précision des capteurs. Pour adresser ces problèmes et rendre la méthode utilisable, les hyperparamètres α et β sont introduits.

Le paramètre α permet d'adoucir la priorisation. Lorsque $\alpha = 0$, L'échantillonnage se fait suivant la distribution uniforme. Dans ce cas, il n'y a pas de priorisation du

tout. Plus la valeur de α augmente, plus l'apprentissage sur les cas particuliers est encouragé. Lorsque l'entraînement est fait sur les corpus issus du rééchantillonnage du corpus original, le résultat de l'apprentissage est biaisé. Ce biais introduit par la priorisation est corrigé pendant l'entraînement en pondérant les échantillons à l'aide des poids définis dans l'équation 5.2.

$$w_i = \left(\frac{1}{N}\frac{1}{P(i)}\right)^{\beta} \tag{5.2}$$

Le paramètre β, choisi entre 0 et 1, permet de moduler la correction du biais. Lorsque $\beta = 1$ le biais est complètement corrigé aux dépens de la vitesse d'apprentissage. Le paramètre α augmente l'agressivité de la priorisation tandis que le paramètre β augmente la correction associée. Il y a donc un équilibre à trouver entre les deux.

Cette méthode permet d'adresser le problème de déséquilibre de corpus. Cependant, les deux hyperparamètres dont elle dépend sont difficiles à choisir. Il est donc intéressant d'explorer d'autres méthodes de priorisation.

5.2.2 Prioriser les données à l'aide du gradient

Une autre façon de prioriser les échantillons est d'utiliser la limite supérieure du gradient comme expliqué dans l'article Katharopoulos et Fleuret, 2018. Comme son nom l'indique, cette méthode s'appuie sur une approximation de la norme du gradient du réseau pendant son entrainement. Les articles Alain et al. (2015) et Loshchilov et Hutter (2015) montrent que la norme du gradient est une expression de ce que le réseau peut apprendre pour un échantillon. Cependant le calcul de la norme du gradient a un coût prohibitif. Pour échapper à ce problème l'article Katharopoulos et Fleuret, 2018 introduit une estimation de cette norme à la fois précise et facile à calculer. Il s'agit d'utiliser pour cette estimation une limite supérieure de la norme du gradient. Cette méthode a l'intérêt de nécessiter moins d'hyperparamètres que la méthode précédemment exposée. En effet, ici le seul hyperparamètre est la taille du super lot (en anglais *superbatch*). Contrairement au PER qui nécessite de réentraîner itérativement les modèles, la méthode basée sur le gradient actualise l'importance des échantillons au fur et à mesure de l'entraînement, ce qui lui permet d'être plus efficace. L'algorithme 3 montre le pseudo-code de l'implémentation que nous avons utilisée pour cette méthode.

Algorithme 3 Priorisation par le gradient

Entrée: *donnes*, N nombre de pas, *tsl* taille du super lot, *tl* taille du lot
  $donnesDentrainement \leftarrow donnes$
  pour $i = 0$ to N **faire**
    $super\_lot \xleftarrow{\text{tsl}} \mathcal{U}(donnesDentrainement)$
    $g = normesuperieurdugradient(super\_lot)$
    $\mathcal{G} \leftarrow distribution\ from\ g$
    $poids \leftarrow \frac{1}{Bg}$

    $lot \xleftarrow{\text{tl}} \mathcal{G}(super\_lot)$
    $tape\_entranement(lot, poids)$
  fin pour

5.3 Mise en œuvre

L'objectif est de comparer les performances des méthodes d'apprentissage standard avec les performances des méthodes d'apprentissage utilisant la priorisation afin d'évaluer ces techniques. Dans un premier temps, les résultats des prédictions des différents modèles sont comparés. Ensuite, la capacité de ces modèles au sein d'un algorithme de contrôle à commande prédictive est évaluée dans le cas d'un drone et d'un bateau.

5.3.1 Comparaison des modèles en simulation

Dans le chapitre précédent, il a été montré que les réseaux de neurones sont capables de réaliser de l'identification de modèles de façon plus précise que les algorithmes de la famille ARMA en terme de l'erreur de précision à un pas et sur une trajectoire. Pour réaliser cela des corpus riches ont été utilisés. Ces corpus étaient générés de façon à être équilibrés c'est-à-dire contenant de nombreux exemples pour l'ensemble des dynamiques considérées. C'était l'objet du contrôleur simplistique décrit précédemment. Cela implique des limitations importantes pour l'application des réseaux de neurones à l'identification de système. Ici, l'objectif est de démontrer que l'utilisation des algorithmes de priorisation permet d'atténuer ce problème.

Test de prédiction des trajectoires finales d'un drone

Afin d'évaluer les capacités de modélisation des différentes méthodes, les trajectoires obtenues en partant d'une situation initiale connue et en appliquant des commandes constantes sont calculées. Dans ces conditions, il est facile de reproduire la même expérience avec différents modèles. Comme vu précédemment, les modèles que nous utilisons permettent de prédire l'évolution des vitesses du drone. Afin de pouvoir comparer les résultats des trajectoires finales du drone, les prédictions en vitesse sont intégrées pour que la trajectoire du drone en position puisse être tracée. Cela permet d'avoir un résultat visuel facilement interprétable et permet d'avoir une idée des capacités de suivi de trajectoires de ces modèles.

Pour mettre en avant l'intérêt de l'échantillonnage priorisé, les différents modèles sont comparés en termes d'erreurs sur la trajectoire des positions. Chaque modèle est utilisé pour calculer la trajectoire du drone sur 20 pas de temps soit 4 secondes alors qu'une commande constante est appliquée. La commande en vitesse appliquée ($u_x = 0.5\,\mathrm{m/s}$, $u_y = 0.5\,\mathrm{m/s}$, $u_z = 0.1\,\mathrm{m/s}$, $u_{rz} = 1\,\mathrm{rad/s}$) est choisie de sorte que toutes les dimensions de la dynamique soient sollicitées. La visualisation de cette trajectoire est observée dans la figure 5.1. Cette figure montre en rouge la référence, en cyan le modèle ARMA, en vert le modèle obtenu par l'entraînement standard d'un réseau de neurones et en bleu le modèle correspondant à un entraînement priorisé du même réseau.

D'abord, le réseau de neurones est entraîné sur un corpus très déséquilibré. Ce corpus a été généré à l'aide d'un contrôleur PID qui n'explore pas tout l'espace d'action. Ensuite des exemples avec des dynamiques plus riches générés par un controleur explorant l'espace d'action de façon aléatoire (décris précédemment ici 4.4.1) ainsi que les données collectées lors de l'utilisation du contrôleur à commandes prédictives sont progressivement ajoutées. Le modèle est réentraîné sur l'ensemble des données à chaque passe : données déséquilibrées et nouvelles données plus riches. Le corpus est construit de sorte que les premiers entraînements du réseau de neurones n'observent

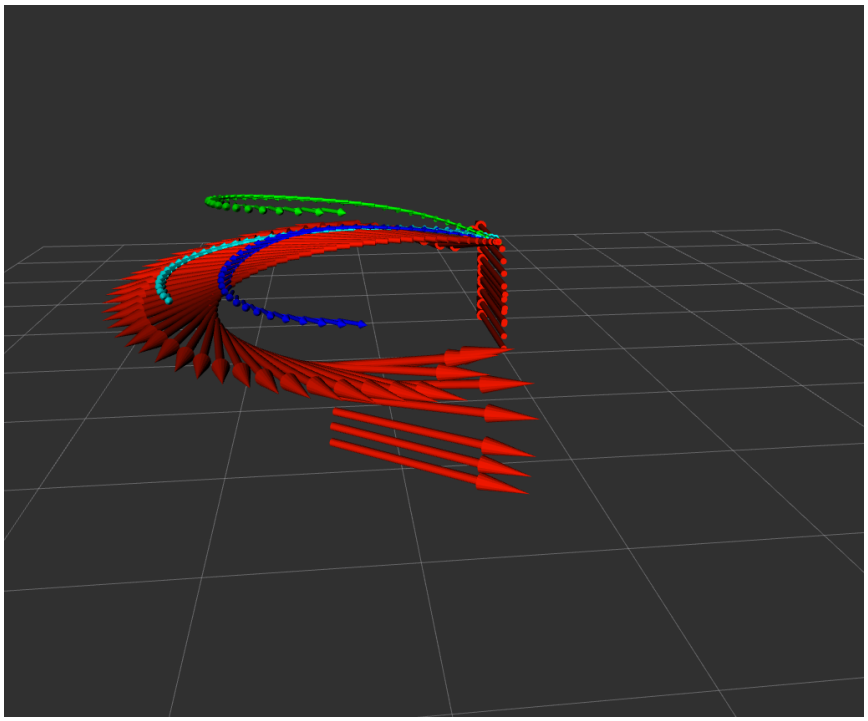


FIGURE 5.1 – Comparaison des simulations de trajectoires obtenues avec différents modèles : ARX (cyan), réseau de neurones (vert) et réseau de neurones avec priorisation (bleu).

que des commandes étant composées de consignes en vitesses suivant les axes x et y. Les entraînements successifs observeront progressivement des échantillons contenant des commandes en rotation et en déplacement vertical. Pour cette expérience, le choix des hyperparamètres est important car il affecte grandement les expériences. Les résultats présentés ont été obtenus avec $\alpha = 0,6$ et $\beta = 0,4$. La figure 5.2 montre la mesure de l'erreur en position correspondant à l'expérience illustrée précédemment en figure 5.1. Le graphique met en avant que le modèle utilisant la priorisation est capable de mieux mettre à profit les nouvelles données. Dans cette expérience, les deux réseaux sont évalués après avoir été entraînés huit fois chacun sur le corpus en expansion. La supériorité du modèle bénéficiant de la priorisation est nettement observée.

Poursuite des résultats en validations croisées

Dans le chapitre précédent les performances d'un modèle utilisant des réseaux de neurones ont été comparées avec celles de la méthode ARMA sur un corpus varié en faisant une validation croisée afin de s'assurer de la robustesse des résultats. Il est naturel de faire la même évaluation avec les modèles utilisant l'apprentissage priorisé. Les résultats sont détaillés dans le tableau 5.1. L'utilisation des méthodes de priorisation apporte une amélioration de 5 à 10 % en erreur à un pas. La méthode de priorisation utilisant le gradient, notée "GRAD" dans la table, est systématiquement meilleure que les réseaux entrainés sans priorisation. Les résultats obtenus avec le PER sont moins constants, ce qui est probablement dû à la complexité du choix des hyperparamètres. En erreur à plusieurs pas, les conclusions sont similaires. Une fois encore l'amélioration des performances est de l'ordre de 5 à 10 %. Une fois de plus les résultats sont obtenus sur des corpus de test variés. De plus, il s'agit de résultats moyennés sur dix expériences. Cela démontre donc l'amélioration des résultats de façon générale.

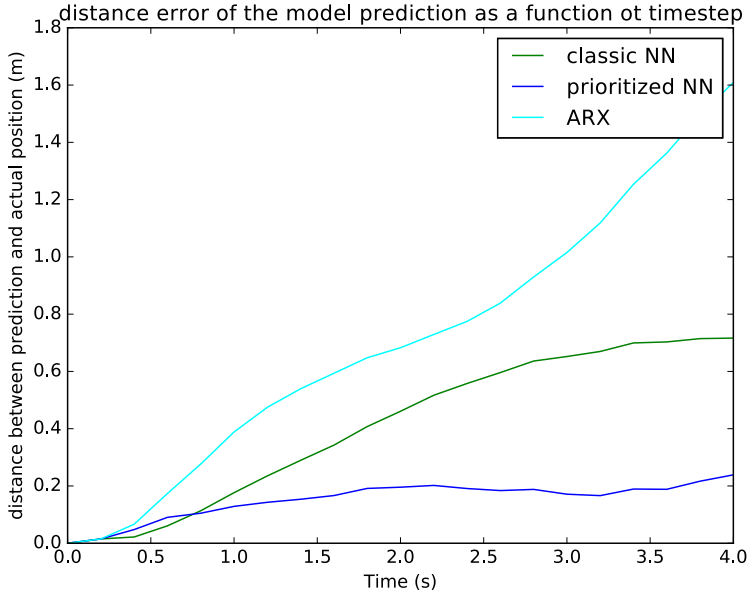


FIGURE 5.2 – comparaison de l'erreur en position obtenue avec différents modèles : ARX (cyan), réseau de neurones (vert) et réseau de neurones avec priorisation (bleu).

Test set	0	1	2	3	4
	Erreur à un pas				
ARMA	0.836	0.873	0.850	0.855	0.883
STD	0.311	0.346	0.109	0.329	0.481
PER	**0.298**	**0.328**	0.122	0.316	0.492
GRAD	**0.298**	0.331	**0.100**	**0.315**	**0.473**
	Erreur pour une trajectoire				
ARMA	1.003	0.986	1.056	0.983	0.958
STD	0.214	**0.263**	0.166	0.257	0.709
PER	0.227	0.271	0.179	0.250	**0.685**
GRAD	**0.202**	0.271	**0.155**	**0.239**	0.705

TABLE 5.1 – Résultat en validation croisée pour le drone simulé.

5.3.2 Comparaison des modèles réels

Cette section s'intéresse au corpus obtenu sur le terrain à l'aide du drone équipé d'un gps RTK comme décrit au chapitre précédent. Les résultats obtenus sont dans le tableau 5.2. Sur le système réel, nos méthodes de modélisation sont mises à l'épreuve. Les plages d'acquisition sont courtes du fait des limitations des batteries. Les conditions météorologiques comme le vent compliquent fortement l'identification du système. Malgré ces conditions difficiles, les approches proposées ne dégradent pas les performances de la modélisation. À l'inverse, de légères améliorations sont observées.

Les modèles ont aussi été testés sur d'autres corpus, colléctées dans d'autres conditions. Le corpus DaISy DE MOOR et al., 1997 est un corpus d'identification de système ouvert. Dans ce corpus deux systèmes étaient pertinents pour cette étude : un bras

Test set	0	1	2	3
	Erreur à un pas			
ARMA	0.778	0.812	0.783	0.804
STD	0.179	**0.549**	0.144	0.116
PER	0.190	0.563	0.162	0.134
GRAD	**0.165**	0.560	**0.137**	**0.108**
	Erreur pour une trajectoire			
ARMA	1.054	1.012	1.046	1.014
STD	0.863	0.738	0.543	0.804
PER	**0.861**	**0.720**	**0.519**	**0.783**
GRAD	0.890	0.728	0.534	0.822

TABLE 5.2 – Résultats de la mesure d'erreur à un pas et d'erreur de trajectoire de la validation croisée pour le drone réel.

robotique flexible et un bras de lecteur de disque. Dans l'ensemble les résultats observés sur le drone se retrouvent dans l'identification de ces systèmes : les approches par apprentissage sont plus performantes que l'approche classique à l'aide d'ARMA et l'utilisation de la prioritisation améliore encore les résultats.

5.3.3 Utilisation des modèles dans un contrôleur à commande prédictive

Pour évaluer la qualité des modèles en terme de contrôle, ils sont utilisés au sein d'un contrôleur à commande prédictive. L'évaluation du contrôleur est faite sur une tâche de suivi de trajectoire. La trajectoire utilisée consiste à suivre les arêtes d'un cube. L'intérêt d'une telle trajectoire est que toutes les dynamiques (v_x, v_y, v_z) sont sollicitées. La trajectoire souhaitée est définie à l'aide d'une cible se déplaçant à vitesse constante. Les modèles précédemment évalués sur leur prédiction de trajectoires sont testés sur cette nouvelle tâche. Pour l'évaluation, la distance entre la cible et le drone est utilisée comme mesure de l'erreur. Les résultats de cette évaluation sont montrés dans le tableau 5.3.

	ARX	***Réseau***	***Réseau avec priorisation***
erreur maximum [m]	1.789	0.882	0.819
aerreur moyenne [m]	1.272	0.562	0.508

TABLE 5.3 – erreur maximum et moyenne

Le réseau de neurones entraîné de façon classique est très supérieur à la méthode ARMA que ce soit en erreur maximum sur l'épisode ou en erreur moyenne. Le réseau entraîné à l'aide de la priorisation améliore encore cette performance en diminuant l'erreur maximum de 8% et l'erreur moyenne de 12%. Il est intéressant de noter que pour cette expérience, les fréquences de contrôle sont fixées à 5 Hz. Il serait possible d'améliorer les performances des modèles ARMA simplement en augmentant la fréquence de contrôle. La seule contrainte dans ce cas est la nécessité d'avoir accès à une mise à jour de l'odométrie assez rapide. En revanche cela est plus difficile pour les réseaux de neurones. En effet, pour pouvoir maintenir une fréquence de contrôle à 5 Hz, il est nécessaire que l'ensemble des calculs de la commande prédictive prenne

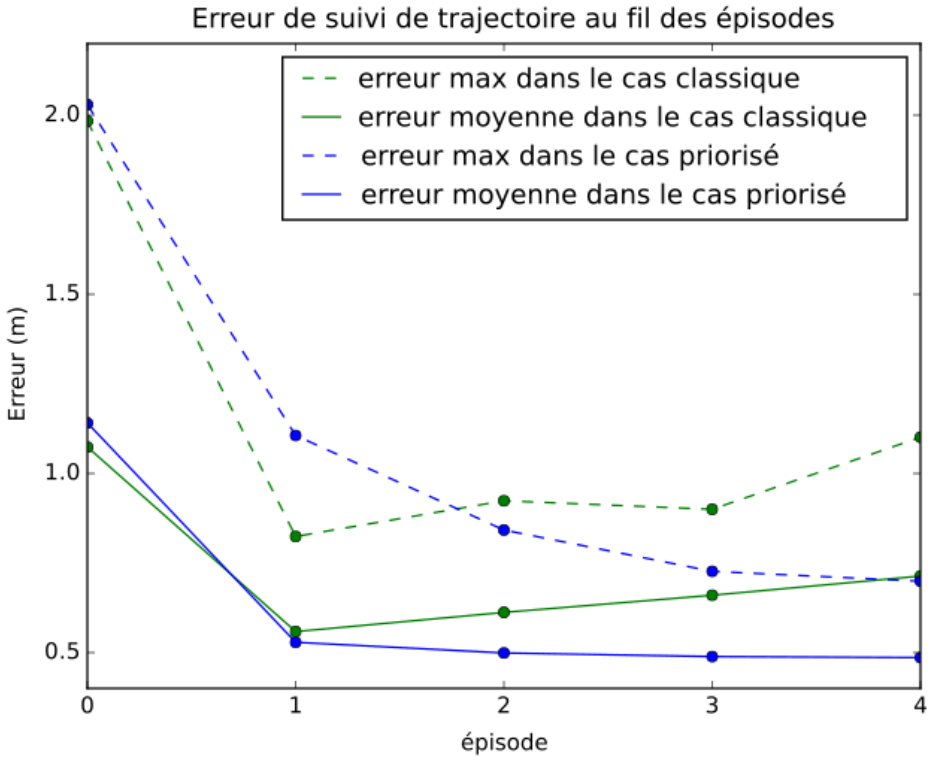


FIGURE 5.3 – Comparaison des performances des modèles priorisé et non priorisé au cours de plusieurs épisodes.

moins que 200 millisecondes, ce qui peut poser problème du fait du coût en temps de calcul associé aux réseaux de neurones. Ce problème de performance est mitigé par les améliorations des technologies matérielles, en particulier l'accessibilité des cartes graphiques embarquées.

L'avantage des réseaux de neurones est leur capacité à s'améliorer lorsque de nouvelles données sont disponibles pour l'entraînement. Afin de comparer les capacités de l'entraînement priorisé et de l'entraînement classique, le contrôleur est évalué sur une tâche au cours de plusieurs épisodes. Chaque épisode représente un essai. Entre chaque épisode le drone se pose et le réseau est entraîné à nouveau sur le corpus combinant les données initiales et les données collectées lors des épisodes précédents. Le résultat de l'expérience est montré dans la figure 5.3. Le réseau de neurones utilisant le rééchantillonnage priorisé est capable de meilleures performances à la fois en termes d'erreurs maximales et moyennes à mesure qu'il s'entraîne. À mesure que le corpus augmente grâce aux données collectées en remplissant la tâche, il devient plus déséquilibré. En effet la tâche est constituée de beaucoup de lignes droites et seulement de quelques changements de direction. Cela a une influence négative sur l'entraînement classique mais pas sur la version priorisée.

Le principal avantage des réseaux de neurones est leur capacité à modéliser des systèmes très complexes. Dans l'expérience précédente, le modèle implémenté inclut le contrôleur bas niveau. Par conséquent, le modèle du système dont la commande prédictive a besoin est quasiment linéaire, le contrôleur bas niveau amortissant les non-linéarités du système. Pour tester notre implémentation sur un problème plus

complexe, une masse est suspendue sous le drone. Cela rend la dynamique beaucoup plus compliquée. La masse de notre drone est de 1.477 kg et la masse ajoutée pèse 150 g. La distance entre le drone et la masse est constante. La masse est contrainte à rester dans un cône de $\frac{\pi}{3}$ rad en dessous du drone. Ces conditions permettent de limiter l'instabilité du drone. Afin d'être capable de modéliser le système, l'historique utilisé est allongé. Jusqu'alors, seulement les deux pas précédents étaient considérés (i.e 400 millisecondes d'historique). Pour pouvoir capturer l'effet de la masse, cet historique est porté à 10 pas, soit 2 secondes d'historique. Dans un premier temps, le modèle ARMA et le réseau de neurones sans priorisation sont comparés. La table ?? montre que les deux modèles ont des difficultés. Mais le contrôleur utilisant des réseaux de neurones reste meilleur.

	ARX	***Réseau de neurones***
erreur max [m]	2.250	1.536
erreur moyenne [m]	1.220	0.946

TABLE 5.4 – erreur maximum et erreur moyenne

Pour évaluer l'intérêt de réentraîner le réseau de neurones à l'aide de la méthode de priorisation, la même expérience que précédemment est faite. Les résultats sont montrés dans la figure 5.4. Dans ce cas, aucun des entraînements ne donne de meilleurs résultats. Plusieurs facteurs peuvent expliquer cela. Dans un premier temps la dynamique étant plus complexe, une amélioration de l'architecture des réseaux de neurones pourrait être nécessaire. D'autre part, pour cette expérience, les hyperparamètres de la priorisation, α et β, ont été laissés inchangés vis-à-vis de l'expérience précédente. Il serait nécessaire d'effectuer une nouvelle recherche pour trouver leur valeur. Enfin, le contrôleur bas niveau est calibré pour le drone normal et n'a pas été modifié lors de l'ajout de la masse. Il est possible que l'interaction entre la masse et le contrôleur rende la dynamique beaucoup plus stochastique que déterministe, ce qui rend l'identification de système particulièrement complexe et nécessiterait des méthodes de modélisation différentes. De plus l'angle de la masse suspendue est inconnu, notre état est donc seulement partiellement observé et l'identification en est grandement désavantagée.

Comme dans le chapitre précédent, il semble pertinent de s'intéresser à un autre système afin de pouvoir juger de la capacité de notre méthode à produire des modèles pour des systèmes variés. Les expériences de suivis de pistes à l'aide d'un bateau commencées au chapitre précédent sont donc reprises en s'intéressant cette fois-ci aux performances des modèles priorisés. Afin de pouvoir discriminer la qualité des différents modèles, ils sont évalués sur des tâches de plus en plus compliquées. La complexité des différentes tâches tient dans la géométrie des pistes considérées. Dans un premier temps l'algorithme est testé sur une piste composée de lignes droites et d'arcs de cercle présentant donc un changement de courbures à plusieurs endroits. Ensuite, l'expérience est faite avec une piste carrée dont les angles à 90 degrés présentent un défi majeur pour le bateau.

La principale difficulté de la piste composite est la discontinuité de sa courbure aux endroits où les lignes droites deviennent des arcs de cercle. La figure 5.5 montre comment les différentes méthodes d'apprentissage se comportent après 670 pas de temps d'une durée de 0.2 seconde (soit un total de 2 minutes et 14 secondes). Les

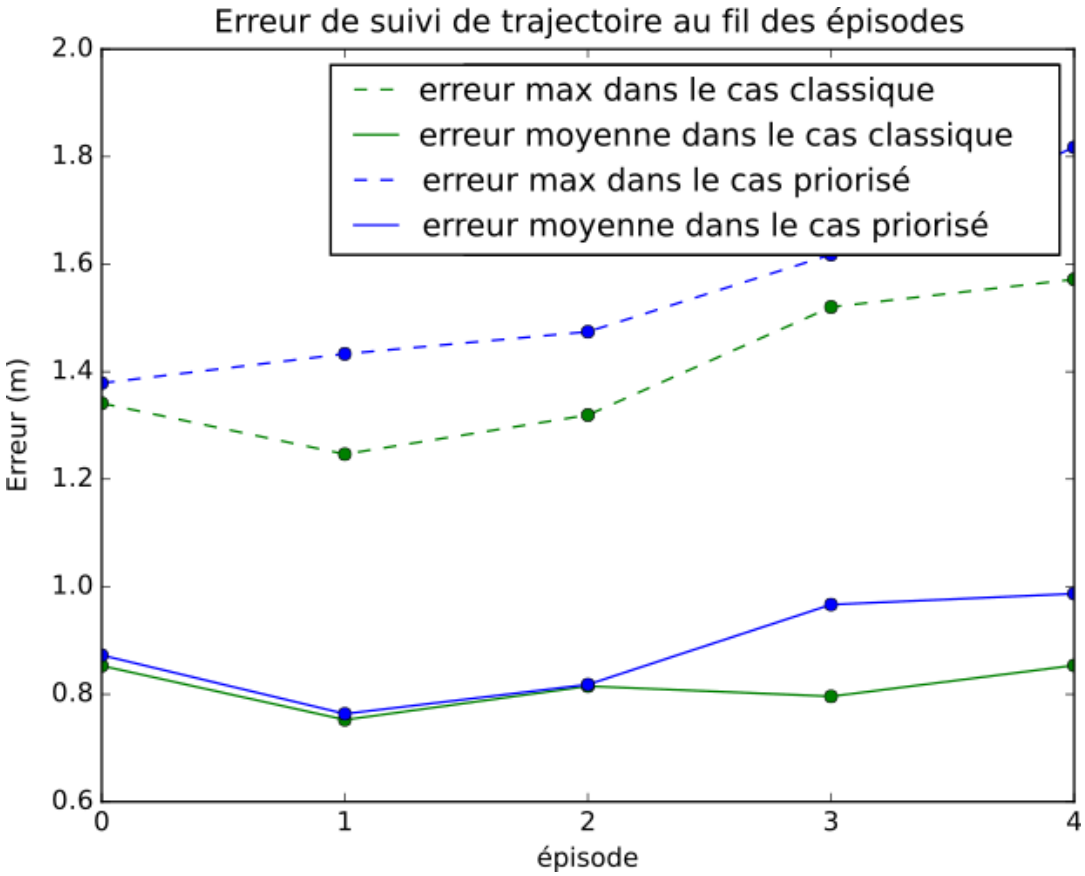


FIGURE 5.4 – Évaluation des modèles avec le drone déséquilibré.

modèles montrés sont les meileurs pour chaque catégorie. Pour le PER le meilleur résultat est obtenu pour $\alpha = 0.7$ et $\beta = 0.1$ alors que le meilleur modèle basé sur le gradient est obtenu pour une taille de lot de 768.

Dans ce cas il semble que les modèles priorisés soient meilleurs que celui utilisant une méthode d'entraînement standard. En effet, ce dernier effectue un dépassement important lorsqu'il rejoint la piste et a du mal à garder l'allure des autres réseaux. Le modèle basé sur la priorisation à l'aide du gradient effectue un dépassement au premier changement de courbes aux coordonnées $x = 20; y = 4$ mais réussit à suivre la piste et bat les autres modèles à la course. Enfin le modèle basé sur l'entraînement PER réussit à suivre la piste et suit le modèle basé sur le gradient de près.

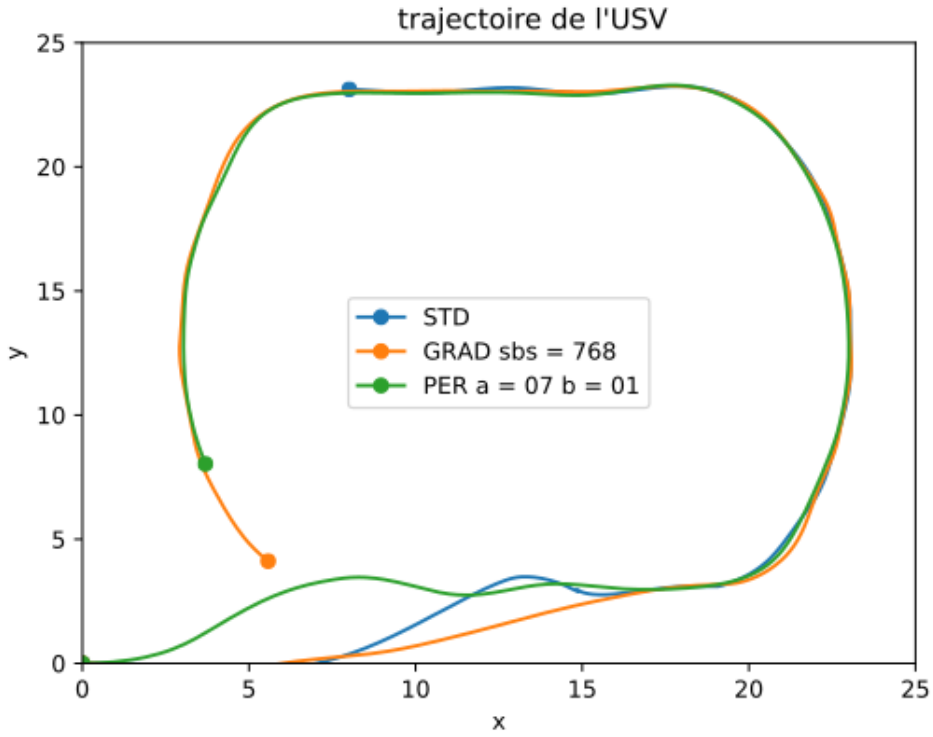


FIGURE 5.5 – Comparaison des modèles entraînés de façon standard (STD) et priorisées (PER/GRAD) sur la carte composite

La piste suivante est beaucoup plus complexe à suivre pour le bateau. En effet ses angle à 90 degrés requièrt au contrôleur plus d'anticipation. Malgré les difficultés les différents modèles réussissent tous à suivre la piste avec l'aide d'un terme de coût correspondant à la direction du bateau. En effet sans ce terme le bateau reste coincé à l'intérieur des angles, l'ajout de l'information du sens de la piste permet à l'algorithme de contrôle de favoriser les trajectoires tournant dans le sens correct et de s'assurer que des trajectoires propices sont toujours évaluées. La figure 5.6 montre les trajectoires suivies par les différents modèles lors de leurs meilleurs essais. Dans ce cas, le modèle PER est le plus rapide mais celui utilisant le gradient est le plus respectueux de la piste, obtenant le meilleur score en terme de coût de carte. Le modèle basé sur le gradient obtient un score moyen pour le coût correspondant à la carte de 3,9 et un coût global de 20, le PER obtient un coût moyen pour la carte de 6,9 et un coût global

de 18,7 tandis que le modèle entrainé classiquement obtient un coût de carte de 6,7 et un coût globale de 22,3.

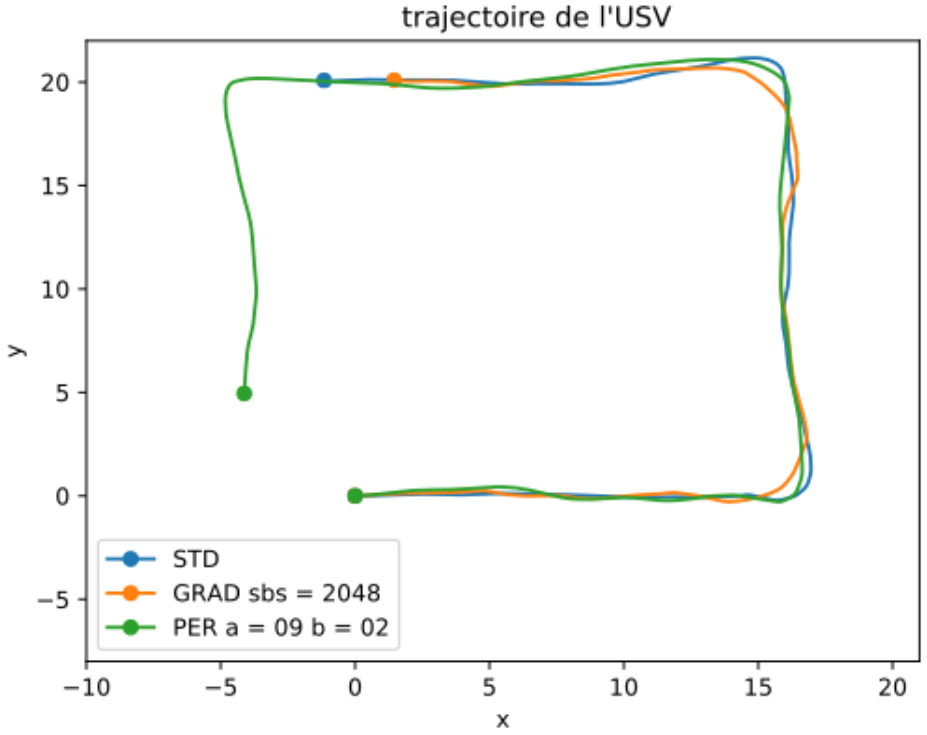


FIGURE 5.6 – Comparaison des modèles entraînés de façon standard (STD) et priorisées (PER/GRAD) sur la carte carrée.

Afin d'étudier la pertinence des approches priorisées, leur performance est analysée au cours de 15 épisodes. Cela permet d'avoir une idée de la robustesse des différents modèles. De même que dans le chapitre 3 on s'interesse à la variabilité des performances suivant les paramètres utilisés par l'algorithme de contrôle. En effet, il a été discuté que les paramètres de la simulation de trajectoire pour le calcul de la commande ont une grande importance. En particulier le nombre de trajectoires de commandes échantillonnées (nombre d'échantillons dans la suite), l'horizon des trajectoires simulées ainsi que l'écart-type utilisé lors de l'échantillonnage ont un impact important. L'impact de ces différents paramètres est comparé ici pour des modèles utilisant les priorisations PER et gradient. Les résultats avec des modèles non priorisés sont montrés pour comparaison.

La figure 5.7 montre comment la moyenne du coût de la carte évolue avec le nombre d'échantillons utilisés dans le *Model Predictive Path Integral* (MPPI). Dans le cas présenté, utiliser moins de 500 échantillons rend le contrôleur très instable. À l'inverse au-delà de 4000 échantillons les limites des capacités de notre algorithme en python se font sentir et au-delà de 6000 échantillons, le contrôleur ne peut plus fonctionner correctement du fait du délai entre les observations et le résultat du calcul de la trajectoire optimale. Le point à observer ici est le fait qu'en moyenne les modèles priorisés performent mieux que le modèle de référence. L'entraînement utilisant le gradient en particulier est très bon dans son suivi de la piste.

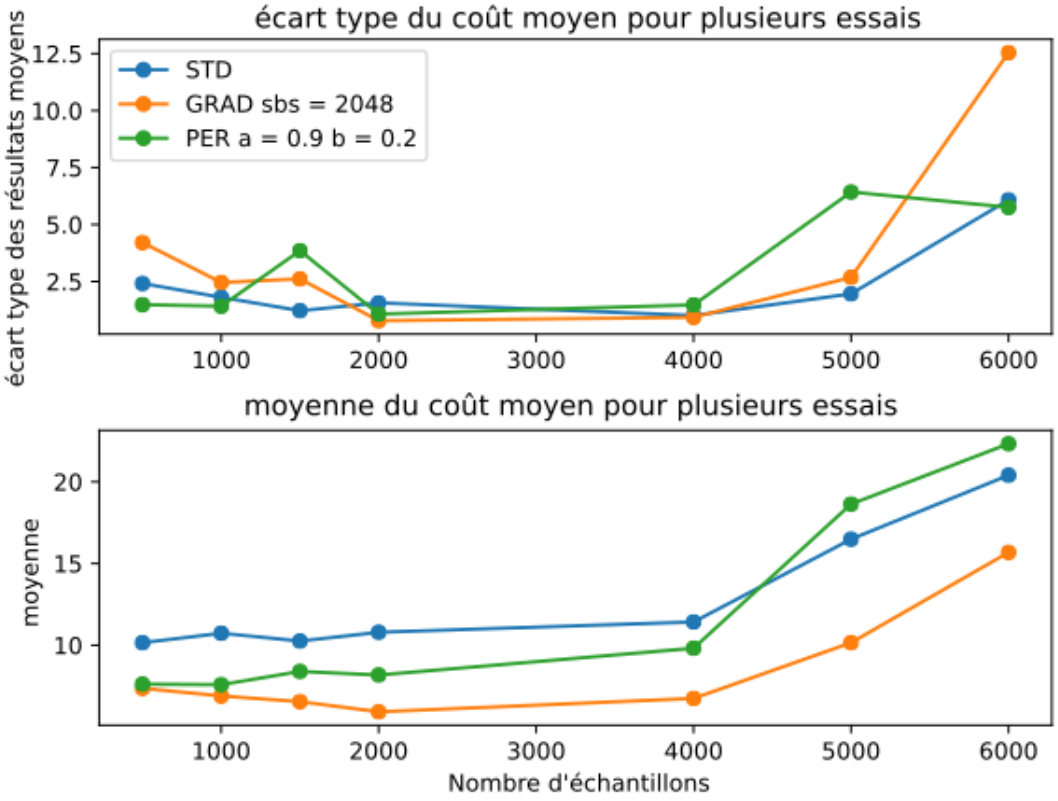


FIGURE 5.7 – Coût associé à la carte moyenné sur plusieurs essais pour un nombre d'échantillons croissant.

La figure 5.8 montre l'influence de l'horizon des trajectoire évaluées lors de la recherche effectuée par l'algorithme. Pour un horizon inférieur à 5 pas les trajectoires sont trop courtes pour permettre la prise en compte de la dynamique du bateau. Au delà de 40 pas on se heurte une fois de plus à la limitation en terme de temps de calcul. D'autre part il est intéressant de noter que plus les trajectoires sont longues, plus l'espace des actions possibles est grand et donc il est souhaitable d'augmenter le nombre d'échantillons évalués lorsque le l'horizon des trajectoires augmente. Cela implique une contrainte encore plus forte en terme de puissance de calcul. De plus, plus les trajectoires sont longues plus les erreurs de modélisation s'accumulent, ajoutant à l'imprécision de l'évaluation des trajectoires.

La figure 5.9 montre l'influence de la variance de l'échantillonnage des commandes à évaluer. Là encore, il y a dans l'ensemble une nette supériorité des modèles priorisés à l'exception du gradient qui pour une variance de 0.9 rencontre des difficultés. En effet, un des 15 épisodes est un total échec et impacte négativement son résultat moyen.

Les paramètres de l'algorithme de commandes prédictives ne sont pas les seuls à étudier. En effet, il a été vu précédement que les algorithmes de priorisation ont aussi un certains nombre de paramètres affectant leur performance. Pour ce faire une recherche systématique de ces paramètres est effectuée. Le résultat de ses recherches, pour le PER de même que pour la méthode basée sur l'approximation du gradient sont présentés ici.

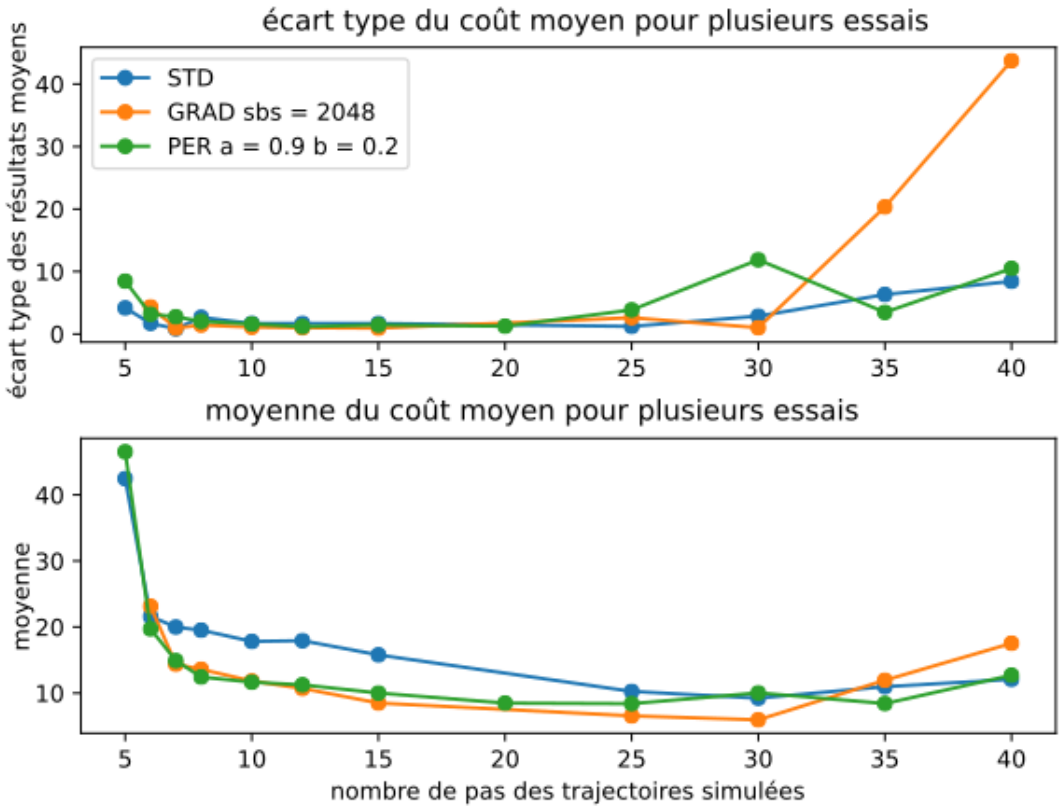


FIGURE 5.8 – Coût associé à la carte moyenné sur plusieurs essais pour un horizon croissant.

L'évaluation est fait sur deux corpus différents : l'un est compris de données générées aléatoirement, constituant ainsi un corpus équilibré, le second corpus est à l'inverse intentionnellement déséquilibré. Les résultats présentés dans les tables sont le résultat moyen sur cinq entraînements différents.

Les figures 5.10 et 5.11 montrent les résultats des recherches pour la méthode par limite supérieure du gradient et le PER sur le corpus déséquilibré. Les figures 5.12 et 5.13 montrent les résultats de la recherche pour le corpus équilibré.

La figure 5.10 montre que les meilleurs résultats de l'entraînement utilisant la priorisation PER sont obtenus lorsque α et β sont tous les deux élevés. Cela signifie que pour obtenir les meilleurs résultats il est nécessaire d'insister sur les éléments les plus difficiles du corpus. Il s'agit de l'effet du α élevé. De plus il est nécessaire de compenser autant que possible pour le biais introduit, via un β élevé. De plus, des valeurs plus élévées de β produisent de meilleurs résultats alors que le choix sur α semble moins impactant. La figure 5.12 montre que pour le corpus plus équilibré la prédiction à un pas est plus homogène, à l'exception de la combinaison de faible β et grand α dont les résultats sont moins bons. Il est intéressant de remarquer que la précision à plusieurs pas sur le corpus équilibré, décrite dans la figure 5.12, est similaire à la précision à plusieurs pas sur le corpus équilibré montré en figure 5.10. Cela montre que la méthode PER améliore les résultats à plusieurs pas en général, ce qui est confirmé par les résultats montrés dans la table 5.5.

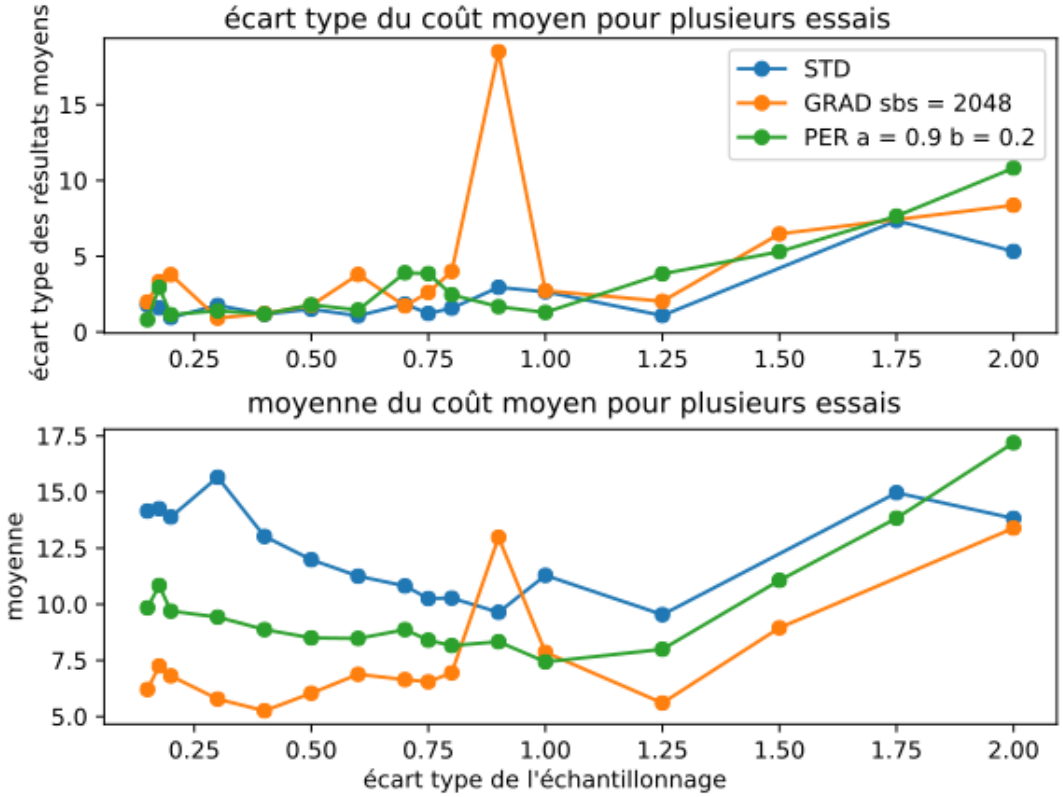


FIGURE 5.9 – Coût associé à la carte moyenné sur plusieurs essais pour une variance de l'échantillonnage croissante.

La figure 5.11 s'intéresse aux résultats de la méthode de priorisation utilisant la borne supérieure du gradient pour le corpus déséquilibré. Lorsque la taille du super lot (*superbatch* en anglais) augmente les performances en termes de précision à un pas s'améliorent. Cependant, l'erreur à plusieurs pas plafonne lorsque la taille du super lot dépasse 2048 échantillons.

	corpus équilibré				corpus déséquilibré			
	single-step RMSE		multi-step RMSE		single-step RMSE		multi-step RMSE	
	mean	std_dev	mean	std_dev	mean	std_dev	mean	std_dev
PER	0.060	**0.006**	0.18	**0.021**	**0.068**	0.0048	0.26	0.088
GRAD	**0.052**	0.007	**0.17**	0.024	0.070	0.0029	**0.25**	**0.024**
STD	0.066	0.014	0.27	0.101	0.082	**0.0028**	0.40	0.1

TABLE 5.5 – Performance des différents réseaux de neurones. Les réseaux utilisant les méthodes PER et gradient sont choisis en selectionnant les paramètres donnant de eilleurs résultats en précision à plusieurs pas. Les valeurs plus faibles indiquent de meilleures performances.

5.3.4 Sur le terrain

L'algorithme démontré en simulation a été adapté par l'équipe pendant la rédaction du manuscrit. Se basant sur ces expériences préliminaires, Antoine Richard et Stéphanie Aravecchia ont poursuivi l'intégration des algorithmes sur la pateforme héron. Ce qui a permis d'aboutir à l'utilisation d'un modèle entraîné à l'aide de la méthode PER

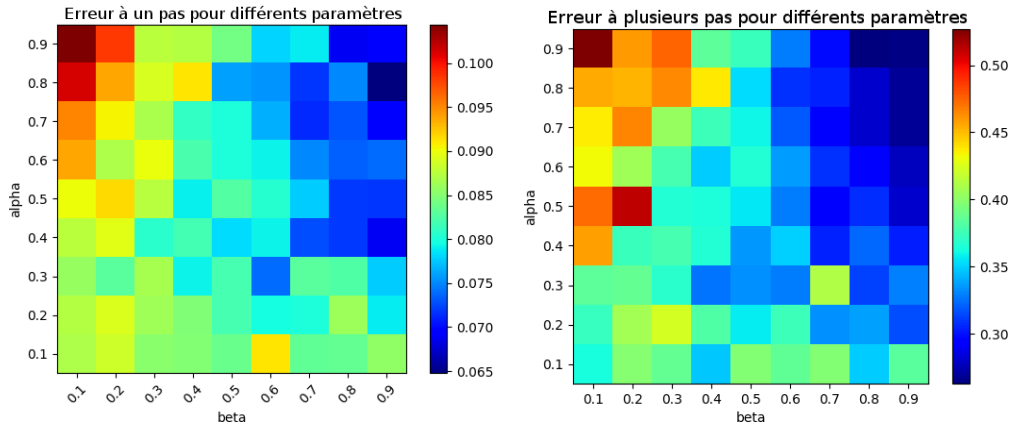


FIGURE 5.10 – Résultats de la méthode PER sur le corpus déséquilibré. À gauche : précision à un pas. À droite : précision à plusieurs pas. Les couleurs plus froides représentent une plus faible RMSE, ce qui est recherché.

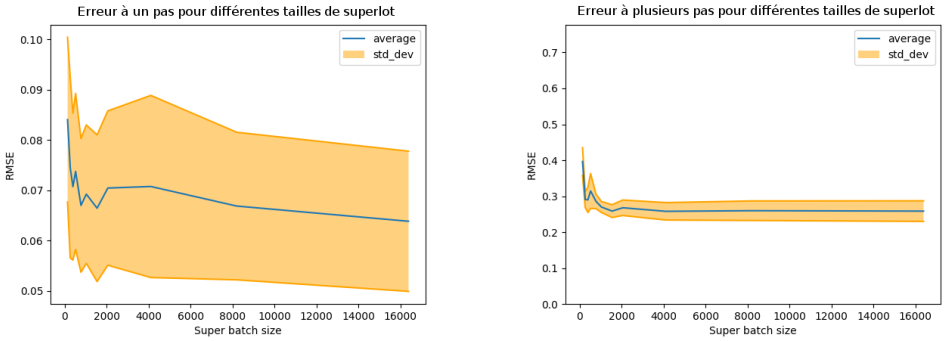


FIGURE 5.11 – Résultats pour la méthode basée sur le gradient pour le corpus déséquilibré. À gauche : précision à un pas. À droite : précision à plusieurs pas. Plus la RMSE est faible meilleur est le résultat. Le couloir orange représente la déviation standard, plus il est étroit plus les résultats sont constants, ce qui est préférable.

au sein d'un algorithme à commande prédictive capable d'effectuer le suivi du bord du lac Symphonie 5.14. La carte de coût utilisée dans ces expériences est construite à partir des données récupérées par le laser monté sur le bateau. Ce dernier permet de détecter la berge ainsi que les obstacles telles que les branches mortes et les racines des arbres. Un couloir de navigation suivant la berge est créé à partir de ces données et le contrôleur MPPI est utilisé pour calculer les commandes envoyées au bateau.

Après que l'algorithme a été testé avec succès sur une simulation du lac, il est testé en conditions réelles. Le problème majeur rencontré pendant les tests réels sont les lignes de pêche qui sont invisibles au laser. En dehors de cela, les principales difficultés rencontrées ont été associées à des branches vues trop tard, ou à des canards et des cygnes qui ont forcé le bateau à faire des demi-tours. Dans l'ensemble, l'algorithme a correctement fonctionné, suivant la berge à la distance désirée sans jamais la percuter. Le robot a réussi à parcourir approximativement 10.2 km en autonomie sans incident. La figure 5.15 montre les résultats obtenus pour la trajectoire et la vitesse du bateau dans différentes conditions météorologiques.

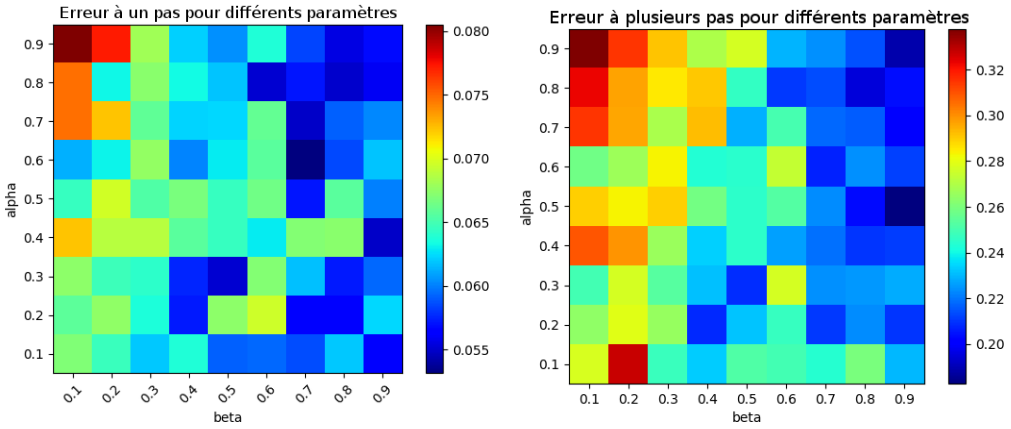


FIGURE 5.12 – Résultats de la méthode PER sur le corpus équilibré. À gauche : précision à un pas. À droite : précision à plusieurs pas. Les couleurs plus froides représentent une plus faible RMSE, ce qui est recherché.

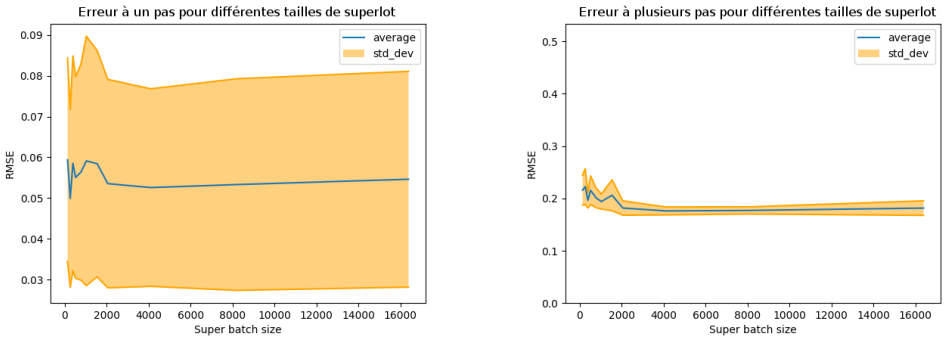


FIGURE 5.13 – Résultats pour la méthode basée sur le gradient pour le corpus équilibré. À gauche : précision à un pas. À droite : précision à plusieurs pas. Plus la RMSE est faible meilleur est le résultat. Le couloir orange représente la déviation standard : plus il est étroit plus les résultats sont constants, ce qui est préférable.

5.4 Conclusion sur la priorisation

Dans ce chapitre l'utilisation de méthode de priorisation lors de l'entraînement de réseaux de neurones a été étudiée. L'intérêt de la priorisation sur des corpus déséquilibrés a été montré. De plus, les performances de l'algorithme PER et de son alternative basée sur l'approximation du gradient ont été comparées. Cependant, il a été remarqué que la gestion des hyperparamètres était un obstacle important lors de l'utilisation de ces méthodes, en particulier dans le cadre du PER. L'intégration de modèle utilisant la priorisation dans un contrôleur à commandes prédictives a permis la réalisation d'un algorithme de suivi de trajectoire dont les performances ont été montrées à la foi en simulation et sur le terrain.

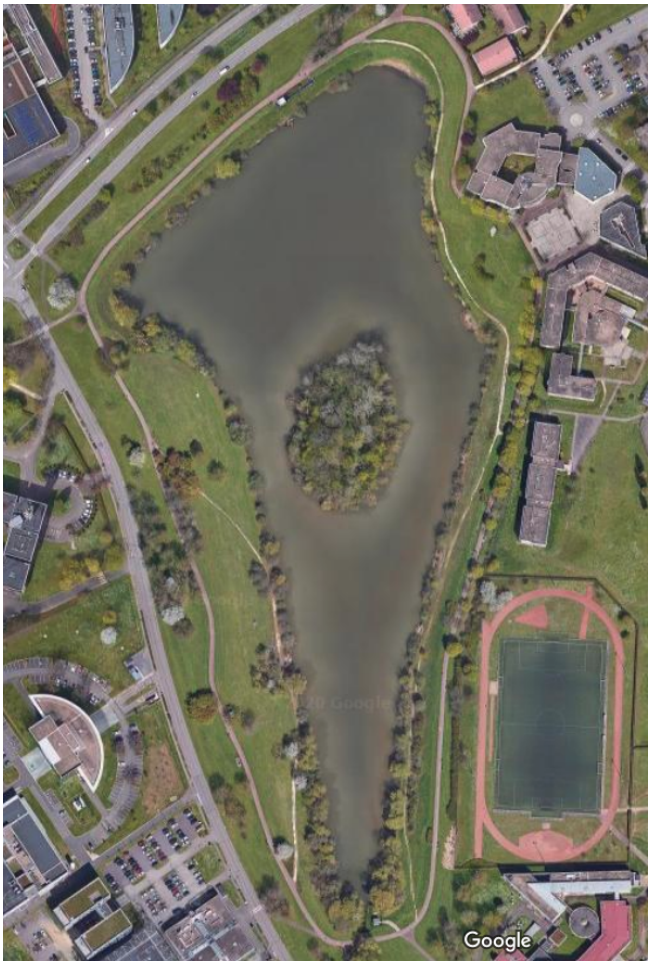
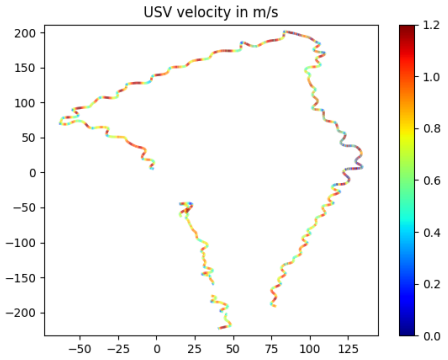


FIGURE 5.14 – Vue aérienne de l'environnement de test, la piste d'athlétisme donne une idée de l'échelle.(Lac Symphonie, 57000 Metz, France, Google Maps, 2020)

Conditions météorologiques idéales



Par temps de vent

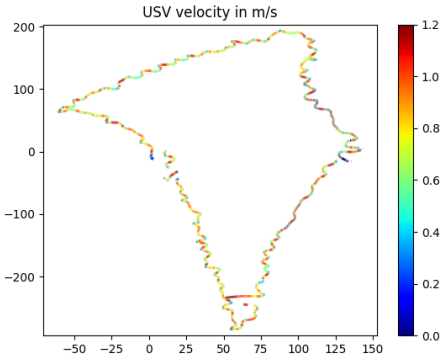


FIGURE 5.15 – Résultats obtenus sur le lac Symphonie. À gauche : Vitesse du robot par temps idéal. À droite : expérience en présence d'un vent moyen de 20.0 hm/h

Chapitre 6

Conclusion et perspectives

Ce chapitre clôt le manuscrit. Dans un premier temps, un résumé du travail présenté y est proposé. Dans un second temps, plusieurs perspectives sont avancées.

6.1 Résumé du travail présenté

Le développement de la robotique mobile permet la réalisation de tâches de plus en plus variées de façon automatisée. Cela représente un avantage majeur dans de nombreuses situations où l'intervention humaine n'est pas possible ou dangereuse. Cela est en particulier vrai pour la robotique aérienne qui est d'une grande utilité dans les tâches d'inspection. Le projet GRoNe dans lequel s'inscrit cette thèse a pour objectif de développer les connaissances et les expérimentations sur ce sujet. Dans ce cadre l'automatisation est un élément clé. Le développement d'algorithmes de contrôle efficaces est une étape importante en ce sens. La réalisation de ces algorithmes requiert la conception de modèles précis des systèmes afin de prédire leur évolution. Ces problématiques de contrôle et de modélisation ont été étudiées.

Dans un premier temps, le chapitre 2 introduit les outils sur lesquels le reste du travail est construit. Après une présentation des enjeux de la robotique mobile, plus précisément dans les contextes aérien et nautique, les outils logiciels utilisés sont introduits. En particulier la structure logicielle ROS permettant d'orchestrer les différents algorithmes collaborant dans un système robotique est présentée, ainsi que le simulateur Gazebo. Ensuite, le contrôle prédictif est introduit. Enfin les méthodes d'identification de systèmes sont présentées, en particulier les avancées en apprentissage automatique dans des champs de recherches proches sont décrites. Notamment sont décrites des architectures de réseaux de neurones couramment employées pour le traitement de séries temporelles, entre autres les architectures ayant rencontré un succès considérable en traitement automatique des langues. L'utilisation de la priorisation lors de l'entraînement de réseaux de neurones est aussi présentée.

Le chapitre 3 s'intéresse aux détails de l'algorithme de commmandes prédictives à intégrale de parcours (MPPI). D'abord, la conception d'une fonction de coût pertinente est discutée. Ensuite le rôle crucial du modèle est expliqué. La capacité de l'algorithme à optimiser une fonction de coût arbitraire sur une trajectoire complète est détaillée. Enfin l'application de cet algorithme est étudiée en simulation sur une tâche de suivi de piste. En particulier, un examen approfondi de l'influence des différents paramètres sur la qualité du suivi est effectué. Il a notamment été établi qu'une augmentation de l'horizon des trajectoires et du nombre de commandes testées améliorait les résultats à condition de rester dans les limites des capacités de calcul du système. De plus, pour l'échantillonnage des commandes à tester, il est nécessaire d'utiliser un écart

type assez grand pour explorer un minimum l'espace des commandes et assez faible pour que le contrôleur puisse converger vers la trajectoire optimale.

Le chapitre 4 détaille la méthode d'apprentissage automatique utilisée pour l'identification de système. Cela inclut la génération des données, l'entraînement des modèles et l'intégration des modèles dans l'algorithme de contrôle. Une étude comparative des différentes architectures de réseaux de neurones est menée. Une implémentation de ces modèles est effectuée dans le cas d'un drone, avec des données de simulation mais aussi avec les données du système réel. De plus, la méthodologie est appliquée à un bateau autonome, en simulation et sur le terrain. Une implémentation pour le bateau est réalisée permettant une grande autonomie puisque l'intégralité des calculs sont faits à bord. Le perceptron multicouche permet d'obtenir des résultats satisfaisants en termes de modélisation ainsi qu'en termes de contrôle une fois intégré dans l'algorithme de contrôle prédictif. Cependant le réseau utilisant le mécanisme d'attention permet d'obtenir desrésultats encore meilleurs en termes de modèle et de contrôle pour un coût en calcul un peu supérieur.

Le chapitre 5 s'intéresse à l'utilisation de méthodes de priorisation pour gérer les problèmes liés aux déséquilibres des données apparaissant dans les corpus collectés sur le terrain. Deux algorithmes de priorisation existant, le PER et la priorisation à l'aide d'une approximation du gradient, sont présentés et comparés. Leur implémentation est réalisée dans différents contextes : en simulation sur un drone, en simulation sur un bateau et en conditions réelles sur le bateau. Les résultats des deux méthodes sont comparables, cependant la méthode utilisant l'approximation du gradient est beaucoup moins coûteuse en termes d'entraînement. À cela s'ajoute la difficulté de sélectionner les bons hyperparamètres de la méthode PER. Les résultats obtenus par ce travail ont permis l'implémentation d'un contrôleur de suivi de berge qui a été testé sur le terrain avec succès.

Le contrôle de robots mobiles est une problématique complexe. Dans ce travail, le problème de la modélisation a été abordée sous l'angle de l'apprentissage automatique. Une étude approfondie de la pertinence des différentes architectures de réseaux couramment utilisés a été menée à la fois sur des données de simulation et sur des données réelles sur deux plateformes aux caractéristiques très différentes : un drone et un bateau. Le déséquilibre des corpus d'entraînement est un problème courant. L'étude de différentes méthodes de prioritisation a mis en avant l'intérêt de ces algorithmes. Les modèles ainsi entrainés ont été évalués au sein d'un contrôleur à commandes prédictives, appliqué sur des tâches de suivi de piste.

6.2 Perspectives

Il existe des cas où les méthodes démontrées ici atteignent leurs limites. En particulier, pour le drone lorsque les conditions météorologiques ne sont pas idéales. L'effet du vent notamment n'est pas du tout pris en compte. Il serait pertinent de s'intéresser à des méthodes d'apprentissage capables de prendre en compte les composantes stochastiques de l'environnement afin d'être plus à même de s'assurer de la robustesse du contrôle dans ces situations.

Une piste de travait ayant un potentiel intéressant serait le transfert de l'apprentissage depuis la simulation vers les conditions réelles. En particuliier, commencer

l'entraînement dans une situation simulée et l'affiner avec des données réelles devrait permettre l'entraînement de modèles performants avec un minimum de données réelles.

Un autre aspect de ce travail où des améliorations notables pouraient être obtenues est l'efficacité des programmes de calcul. Dans ce travail, les algorithmes de contrôle ont été réalisés en python, utilisant au mieux l'efficacité de la bibliothèque numpy. Les réseaux de neurones ont été implémentés en utilisant tensorflow et les capacités de calcul de la carte graphique embarquée. Cependant il serait pertinent d'étudier la possibilité de recompiler les architectures neuronales spécialement pour le matériel à notre disposition, en utilisant des outils tels que tensorRT. Cela devrait mener à des gains significatifs, ce qui permettrait de choisir de meilleurs paramètres pour l'algorithme de contrôle : horizon plus lointain et plus grand nombre de trajectoires testées notamment.

Bibliographie

Abadi, Martín et al. (2016). « Tensorflow : A system for large-scale machine learning ». In : *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, p. 265-283.

Alain, Guillaume et al. (2015). « Variance reduction in SGD by distributed importance sampling ». In : *arXiv preprint arXiv :1511.06481*.

Alsalam, Bilal Hazim Younus et al. (2017). « Autonomous UAV with vision based on-board decision making for remote sensing and precision agriculture ». In : *2017 IEEE Aerospace Conference*. IEEE, p. 1-12.

Amodei, Dario et Jack Clark Greg Brockman Ilya Sutskever Danny Hernandez Girish Sastry (2019). *AI and Compute*. URL : https://openai.com/blog/ai-and-compute/ (visité le 12/07/2020).

Andrychowicz, OpenAI : Marcin et al. (2020). « Learning dexterous in-hand manipulation ». In : *The International Journal of Robotics Research* 39.1, p. 3-20. DOI : 10.1177/0278364919887447. eprint : https://doi.org/10.1177/0278364919887447. URL : https://doi.org/10.1177/0278364919887447.

Angelova, Anelia et al. (2007). « Slip prediction using visual information ». In :

Brown, Tom B. et al. (2020). « Language Models are Few-Shot Learners ». In : arXiv : 2005.14165 [cs.CL].

Browne, Cameron B et al. (2012). « A survey of monte carlo tree search methods ». In : *IEEE Transactions on Computational Intelligence and AI in games* 4.1, p. 1-43.

Che, Zhengping et al. (2018). « Recurrent neural networks for multivariate time series with missing values ». In : *Scientific reports* 8.1, p. 6085.

Cho, Kyunghyun et al. (2014). « Learning phrase representations using RNN encoder-decoder for statistical machine translation ». In : *arXiv preprint arXiv :1406.1078*.

Chollet, François et al. (2015). *Keras*. https://keras.io.

De Moor, Bart et al. (1997). « DAISY : A database for identification of systems ». In : *JOURNAL A* 38, p. 4-5.

Dentler, Jan et al. (2016). « A tracking error control approach for model predictive position control of a quadrotor with time varying reference ». In : *Robotics and Biomimetics (ROBIO), 2016 IEEE International Conference on*. IEEE, p. 2051-2056.

Devlin, Jacob et al. (2018). « Bert : Pre-training of deep bidirectional transformers for language understanding ». In : *arXiv preprint arXiv :1810.04805*.

Dunbabin, Matthew, Alistair Grinham et James Udy (2009). « An autonomous surface vehicle for water quality monitoring ». In : *Australasian conference on robotics and automation (ACRA)*. Citeseer, p. 2-4.

Elkady, Ayssam et Tarek Sobh (2012). « Robotics middleware : A comprehensive literature survey and attribute-based bibliography ». In : *Journal of Robotics* 2012.

Friedman, Jerome, Trevor Hastie et Robert Tibshirani (2001). *The elements of statistical learning*. T. 1. 10. Springer series in statistics New York.

Gonzalez, Jesús et Wen Yu (2018). « Non-linear system modeling using LSTM neural networks ». In : *IFAC-PapersOnLine* 51.13, p. 485-489.

Goodfellow, Ian, Yoshua Bengio et Aaron Courville (2016). *Deep learning*. MIT press.

Griffith, Shane, Georges Chahine et Cédric Pradalier (2017). « Symphony lake dataset ». In : *The International Journal of Robotics Research* 36.11, p. 1151-1158.

Groves, Keir et al. (2019). « MallARD : An Autonomous Aquatic Surface Vehicle for Inspection and Monitoring of Wet Nuclear Storage Facilities ». In : *Robotics* 8.2, p. 47.

Gupte, Shweta, Paul Infant Teenu Mohandas et James M Conrad (2012). « A survey of quadrotor unmanned aerial vehicles ». In : *2012 Proceedings of IEEE Southeastcon*. IEEE, p. 1-6.

Hochreiter, Sepp et Jürgen Schmidhuber (1997). « Long short-term memory ». In : *Neural computation* 9.8, p. 1735-1780.

Hwangbo, Jemin et al. (2017). « Control of a quadrotor with reinforcement learning ». In : *IEEE Robotics and Automation Letters* 2.4, p. 2096-2103.

Katharopoulos, Angelos et François Fleuret (2017). « Biased Importance Sampling for Deep Neural Network Training ». In : *CoRR* abs/1706.00043. arXiv : 1706.00043. url : http://arxiv.org/abs/1706.00043.

— (2018). « Not All Samples Are Created Equal : Deep Learning with Importance Sampling ». In : *CoRR* abs/1803.00942. arXiv : 1803.00942. url : http://arxiv.org/abs/1803.00942.

Khalil, Wisama et Etienne Dombre (2004). *Modeling, identification and control of robots*. Butterworth-Heinemann.

Kingma, Diederik P. et Jimmy Ba (2014). « Adam : A Method for Stochastic Optimization ». In : *CoRR* abs/1412.6980. url : http://arxiv.org/abs/1412.6980.

Koenig, Nathan et Andrew Howard (2004). « Design and use paradigms for gazebo, an open-source multi-robot simulator ». In : *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*. T. 3. IEEE, p. 2149-2154.

Kramer, James et Matthias Scheutz (2007). « Development environments for autonomous mobile robots : A survey ». In : *Autonomous Robots* 22.2, p. 101-132.

LeCun, Yann, Yoshua Bengio et al. (1995). « Convolutional networks for images, speech, and time series ». In : *The handbook of brain theory and neural networks* 3361.10, p. 1995.

Lenain, Roland et al. (2006). « High accuracy path tracking for vehicles in presence of sliding : Application to farm vehicle automatic guidance for agricultural tasks ». In : *Autonomous robots* 21.1, p. 79-97.

Lipton, Zachary C, John Berkowitz et Charles Elkan (2015). « A critical review of recurrent neural networks for sequence learning ». In : *arXiv preprint arXiv :1506.00019*.

Ljung, Lennart (1998). « System identification ». In : *Signal analysis and prediction*. Springer, p. 163-173.

Loshchilov, Ilya et Frank Hutter (2015). « Online batch selection for faster training of neural networks ». In : *arXiv preprint arXiv :1511.06343*.

Lucet, Eric, Roland Lenain et Christophe Grand (2015). « Dynamic path tracking control of a vehicle on slippery terrain ». In : *Control engineering practice* 42, p. 60-73.

Mader, D et al. (2016). « Potential of uav-based laser scanner and multispectral camera data in building inspection. » In : *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences* 41.

Mahé, A., C. Pradalier et M. Geist (2018). « Trajectory-control using deep System Identication and Model Predictive Control for Drone Control under Uncertain

Load. » In : *2018 22nd International Conference on System Theory, Control and Computing (ICSTCC)*, p. 753-758.

Mahé, Antoine et al. (2019). « Importance Sampling for Deep System Identification ». In : *2019 19th International Conference on Advanced Robotics (ICAR)*. IEEE, p. 43-48.

Mahé, Antoine et al. (2020). « Evaluation of prioritized deep system identification on a path following task ».

Mayne, David Q. (2014). « Model predictive control : Recent developments and future promise ». In : *Automatica* 50.12, p. 2967-2986.

Mnih, Volodymyr et al. (2013). « Playing Atari with Deep Reinforcement Learning ». In : *CoRR* abs/1312.5602. URL : http://arxiv.org/abs/1312.5602.

Naegeli, T. et al. (2017). « Real-time Motion Planning for Aerial Videography with Dynamic Obstacle Avoidance and Viewpoint Optimization ». In : *IEEE Robotics and Automation Letters* 2.3, p. 1696-1703. ISSN : 2377-3766. DOI : 10.1109/LRA.2017.2665693.

Narendra, Kumpati S et Snehasis Mukhopadhyay (1992). « Intelligent control using neural networks ». In : *IEEE Control systems magazine* 12.2, p. 11-18.

Narendra, Kumpati S et Kannan Parthasarathy (1992). « Neural networks and dynamical systems ». In : *International Journal of Approximate Reasoning* 6.2, p. 109-131.

Ogunmolu, Olalekan et al. (2016). « Nonlinear systems identification using deep dynamic neural networks ». In : *arXiv preprint arXiv :1610.01439*.

Pêtrès, Clément, Miguel-Angel Romero-Ramirez et Frédéric Plumet (2011). « Reactive path planning for autonomous sailboat ». In : *2011 15th International Conference on Advanced Robotics (ICAR)*. IEEE, p. 112-117.

Pivtoraiko, Mihail, Ross A Knepper et Alonzo Kelly (2009). « Differentially constrained mobile robot motion planning in state lattices ». In : *Journal of Field Robotics* 26.3, p. 308-333.

Pretto, Alberto et al. (2019). « Building an Aerial-Ground Robotics System for Precision Farming ». In : *arXiv preprint arXiv :1911.03098*.

Qin, S Joe et Thomas A Badgwell (2003). « A survey of industrial model predictive control technology ». In : *Control engineering practice* 11.7, p. 733-764.

Quigley, Morgan et al. (2009). « ROS : an open-source Robot Operating System ». In : *ICRA Workshop on Open Source Software*.

Rawlings, J et D.Q. Mayne (jan. 2009). *Model Predictive Control : Theory and Design*.

Richard, Antoine et al. (2020). « Fast Paced Shore Following Using Data Driven Model Predictive Control. »

Rohmer, E., S. P. N. Singh et M. Freese (2013). « CoppeliaSim (formerly V-REP) : a Versatile and Scalable Robot Simulation Framework ». In : *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*.

Schaal, Stefan, Christopher G Atkeson et Sethu Vijayakumar (2000). « Realrning with locally weighted statistical learning-time robot learning with locally weighted statistical learning ». In : *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*. T. 1. IEEE, p. 288-293.

— (2002). « Scalable techniques from nonparametric statistics for real time robot learning ». In : *Applied Intelligence* 17.1, p. 49-60.

Schaul, Tom et al. (2015). « Prioritized experience replay ». In : *arXiv preprint arXiv :1511.05952*.

Silver, David et al. (2016). « Mastering the game of Go with deep neural networks and tree search ». In : *nature* 529.7587, p. 484-489.

Sonnenburg, Christian R et Craig A Woolsey (2013). « Modeling, identification, and control of an unmanned surface vehicle ». In : *Journal of Field Robotics* 30.3, p. 371-398.

Staranowicz, Aaron et Gian Luca Mariottini (2011). « A survey and comparison of commercial and open-source robotic simulator software ». In : *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*, p. 1-8.

Tassa, Y., T. Erez et E. Todorov (2012). « Synthesis and stabilization of complex behaviors through online trajectory optimization ». In : *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, p. 4906-4913.

Turing, A. M. (1950). « Computing machinery and intelligence ». In : *Mind* 59.236, p. 433.

Vaswani, Ashish et al. (2017). « Attention is All you Need ». In : *Advances in Neural Information Processing Systems 30*. Sous la dir. d'I. Guyon et al. Curran Associates, Inc., p. 5998-6008. url : http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

Velasquez, AEB et al. (2020). « Reactive navigation system based on H_∞ control system and LiDAR readings on corn crops ». In : *Precision Agriculture* 21.2, p. 349-368.

Wang, Fei-Yue et al. (2016). « Where does AlphaGo go : From church-turing thesis to AlphaGo thesis and beyond ». In : *IEEE/CAA Journal of Automatica Sinica* 3.2, p. 113-120.

Wehbe, Bilal, Marc Hildebrandt et Frank Kirchner (2017). « Experimental evaluation of various machine learning regression methods for model identification of autonomous underwater vehicles ». In : *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, p. 4885-4890.

Williams, Grady et al. (2016). « Aggressive driving with model predictive path integral control ». In : *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, p. 1433-1440.

Williams, Grady et al. (2017a). « Information theoretic MPC for model-based reinforcement learning ». In : *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, p. 1714-1721.

— (2017b). « Information Theoretic MPC for Model-Based Reinforcement Learning ». In :

Zhang, T. et al. (sept. 2015). « Learning Deep Control Policies for Autonomous Aerial Vehicles with MPC-Guided Policy Search ». In : *ArXiv e-prints*. arXiv : 1509.06791 [cs.LG].

Zhang, Tianhao et al. (2016). « Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search ». In : *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, p. 528-535.