Titre: Title:	Développement d'un système collaboratif de tennis de table basé sur un bras robotique de bureau intégrant l'intelligence artificielle
Auteur: Author:	Baptiste Toussaint
Date:	2025
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Toussaint, B. (2025). Développement d'un système collaboratif de tennis de table basé sur un bras robotique de bureau intégrant l'intelligence artificielle [Thèse de doctorat, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/64787/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/64787/
Directeurs de recherche: Advisors:	Maxime Raison
Programme: Program:	Génie mécanique

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Développement d'un système collaboratif de tennis de table basé sur un bras robotique de bureau intégrant l'intelligence artificielle

TOUSSAINT BAPTISTE

Département de génie mécanique

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*

Génie mécanique

Février 2025

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Cette thèse intitulée :

Développement d'un système collaboratif de tennis de table basé sur un bras robotique de bureau intégrant l'intelligence artificielle

présentée par **Baptiste TOUSSAINT**

en vue de l'obtention du diplôme de *Philosophiæ Doctor*

a été dûment acceptée par le jury d'examen constitué de :

Sofiane ACHICHE, président
Maxime RAISON, membre et directeur de recherche
Abolfazl MOHEBBI, membre
Jean-Philippe ROBERGE, membre externe

REMERCIEMENTS

Je tiens tout d'abord à remercier les membres du jury, les professeurs Sofiane Achiche, Abolfazl Mohebbi et Jean-Philippe Roberge pour leur intérêt et leur disponibilité dans l'évaluation cette thèse. Merci également au professeur Jérôme Le Ny d'avoir accepté de représenter la direction des études supérieures.

Je souhaite évidemment remercier mon directeur de recherche, le professeur Maxime Raison, pour ses nombreux conseils, sa disponibilité et surtout sa confiance inébranlable tout au long de ces quatre années. Grâce à ta confiance, j'ai pu mener à bien cette thèse en conservant un équilibre entre vie académique et personelle. C'est une chance et je t'en suis grandement reconnaissant.

Merci également au fonds de recherche IVADO et à Polytechnique Montréal pour leur soutien financier durant ce projet.

Je souhaite remercier les personnes rencontrées à Montréal : Bastien, Camille, Erika, Hamid, Lauren, Layal, Melika, Nicolas, Sergio, Tarek, Thibault, William, Xavier, et bien d'autres. Je me souviendrai longtemps de tous les bons moments passés ensemble au creux de l'hiver québécois. Un remerciement tout particulier à Marie et Dominic pour leur accueil chaleureux dès mon arrivée à Montréal. Merci pour votre grande générosité et les nombreuses fins de semaine à Saint-Hubert. Une pensée pour mes amis de Cherbourg : Amonda, Anne, Anne-Sophie, Baptiste, Côme, Étienne, Judicaël, Margaux, Thibaut, Valentine, et bien d'autres. Les soirées et les nombreuses heures de tennis m'ont aidé à garder le moral. Merci également à mes amis de longue date, Alexandre, Mathis, Maxime, Quentin, et Nathan, avec qui mon aventure canadienne a commencé en 2019 et sans qui je ne me serais probablement pas lancé dans un doctorat. Merci pour votre amitié précieuse et tous les bons moments passés ensemble, en particulier ces quatre dernières années.

Enfin, je remercie ma famille, mes parents et ma soeur. Vous m'avez donné le goût du travail et la volonté de persévérer dans les défis qui se présentent. Je sais tout ce que je vous dois. Finalement, et évidemment, je souhaite remercier ma femme, Madeleine. Merci pour ton soutien sans faille tout au long de ces quatre ans, en particulier dans les moments plus difficiles. Toi seule sais à quel point ça n'a pas toujours été facile. Je n'aurais pas pu réussir sans toi.

Cette thèse marque la fin d’un chapitre intense de ma vie, débuté en 2019 en tant que jeune étudiant passionné de robotique, commençant un double diplôme à Montréal, et se terminant cinq ans et demi plus tard, en tant qu’ingénieur, docteur et marié. J’ai hâte d’écrire la suite !

RÉSUMÉ

Le marché mondial de la robotique connaît une forte croissance. L'intégration de systèmes robotiques avancés et de l'intelligence artificielle (IA) transforme de nombreux domaines. En particulier, le développement des robots collaboratifs, ou cobots offre de nouvelles opportunités. Cependant, leur coût élevé constitue un frein majeur à leur adoption. Par ailleurs, les robots personnels souffrent de performances limitées. Ils sont moins rapides et moins précis, avec une charge maximale et des espaces de travail restreints, ce qui limite leurs applications.

L'utilisation de l'IA et de l'apprentissage machine augmente les capacités des cobots et robots personnels, ouvrant ainsi de nouvelles opportunités pour les systèmes robotiques. Associée aux nouvelles techniques d'impression 3D, l'IA devrait permettre de réduire l'écart de performance entre les systèmes robotiques personnels abordables et les cobots traditionnels, notamment en matière de dextérité, rapidité et précision.

Le laboratoire dirigé par le professeur Maxime Raison s'est attaqué à la problématique du coût en développant un bras robotique de bureau personnel. Ce robot, équipé d'un préhenseur adaptatif, dispose de cinq degrés de liberté. Il est portable et personnalisable, pour un coût en matériel inférieur à 3,000$ grâce à l'utilisation de l'impression 3D.

D'autre part, le tennis de table est un sujet de recherche en robotique depuis les années 1980. Ce sport constitue en effet un défi de taille pour les chercheurs en robotique en raison des exigences élevées en matière de dextérité d'agilité, de rapidité et de précision nécessaires pour prédire et intercepter les trajectoires de la balle. Ces exigences en font un excellent banc d'essai, régulièrement utilisé pour tester et démontrer les performances des nouvelles technologies.

L'objectif de cette thèse est de développer un système collaboratif de tennis de table, basé sur le prototype du laboratoire du professeur Maxime Raison afin de démontrer les performances atteignables par des bras robotiques de bureau, intégrant l'intelligence artificielle.

Tout d'abord, une méthode combinant les deux approches classiques - apprentissage machine et modèle dynamique – a été développée pour la prédiction de la trajectoire de la balle. Elle a permis d'augmenter la précision de la prédiction de 9% par rapport aux autoencodeurs « Gated-Recurrent Unit » (GRU-TAE) de la littérature. Ensuite, une structure de contrôle a été conçue pour le suivi de trajectoires rapides à haute dynamique, sans nécessiter de connaissance préalable sur le modèle

dynamique du robot. La structure développée permet d'obtenir une erreur moyenne de suivi inférieur à 2° dans l'ensemble de l'espace de travail. Elle peut être appliquée à des systèmes différents pour des applications variées en suivant les guidelines fournies. Enfin, le système complet capable de jouer au tennis de table a été implémenté autour du prototype du laboratoire du professeur Maxime Raison, en intégrant les méthodes les plus performantes identifiées dans la littérature. Le système final est capable de réaliser des échanges avec un utilisateur collaboratif, en renvoyant jusqu'à 81.4% des balles grâce à une optimisation simultanée de la position de la base du robot et du point visé lors du retour. Si ces résultats se rapprochent de ceux obtenus dans la littérature avec des cobots traditionnels (88%), les travaux menés ont également mis en lumière plusieurs limitations liées à l'utilisation d'un bras robotique de bureau. La robustesse et les performances du système sont notamment affectées par la grande difficulté de réaliser des trajectoires rectilignes avec la raquette et les vitesses et couples maximaux des moteurs.

En conclusion, ce système représente une étape supplémentaire dans la démocratisation des bras robotiques personnels et de bureau. Les travaux menés ont montré que l'intégration de l'apprentissage machine, en complément des méthodes traditionnelles basées sur les modèles dynamiques, permet d'améliorer les performances des robots personnels. Les perspectives de cette recherche incluent l'utilisation d'une architecture physique spécifiquement optimisée pour le tennis de table, permettant de réaliser plus facilement les mouvements rectilignes ainsi que l'extension des applications des bras robotiques de bureau à d'autres domaines, tout en continuant à accroitre leurs performances.

ABSTRACT

The global robotics market is experiencing significant growth. The integration of advanced robotic systems and artificial intelligence (AI) is transforming numerous fields. In particular, the development of collaborative robots, or cobots, offers new opportunities. However, their high cost is a major barrier to widespread adoption. Meanwhile, personal robots suffer from limited performance. They are slower and less accurate, with restricted payload capacities and workspaces, which restricts their applications.

The use of AI and machine learning is enhancing the capabilities of cobots and personal robots, opening up new possibilities for robotic systems. Combined with emerging 3D printing techniques, AI is expected to narrow the performance gap between affordable personal robotic systems and traditional cobots, particularly in terms of dexterity, speed, and precision.

The laboratory led by Professor Maxime Raison has addressed the cost issue by developing a personal desktop robotic arm. This robot, equipped with an adaptive gripper, has five degrees of freedom. It is portable and customizable, with a material cost under $3,000 thanks to 3D printing.

Furthermore, table tennis has been a topic of robotics research since the 1980s. This sport presents a significant challenge for robotics researchers due to the high demands for dexterity, agility, speed, and precision required to predict and intercept the ball's trajectory. These demands make it an excellent testing ground, frequently used to evaluate and demonstrate new technological capabilities.

The goal of this thesis is to develop a collaborative table tennis system based on the prototype from Professor Maxime Raison's laboratory to demonstrate the achievable performance of personal desktop robotic arms integrating AI.

First, a method combining two classic approaches—machine learning and dynamic modeling—was developed for ball trajectory prediction. This method increased prediction accuracy by 9% compared to Gated-Recurrent Unit Autoencoder (GRU-TAE) methods from the literature. Then, a control structure was designed for tracking fast, high-dynamic trajectories without any prior knowledge of the robot’s dynamic model. The developed structure achieves an average tracking error below 2° across the entire workspace. It can be applied to different systems for various applications by following the guidelines provided.

Finally, the complete system capable of playing table tennis was implemented using the laboratory prototype, integrating the most effective methods identified in the literature. The final system can engage in rallies with a collaborative user, returning up to 81.4% of shots by optimizing the robot base position and the target point during returns. While these results approach those obtained in the literature with traditional cobots (88%), the conducted work also highlighted several limitations related to using a desktop robotic arm. The system robustness and performance are notably affected by the difficulty of executing straight-line trajectories with the racket, as well as the motors maximum speeds and torques.

In conclusion, the developed system represents a step forward integrating the most effective methods identified in the literature. The work demonstrated that integrating machine learning methods, alongside traditional model-based methods using dynamic equations, enhances the performance of personal robots. Future research prospects include designing a physical architecture specifically optimized for table tennis, facilitating linear movements, and extending the applications of desktop robotic arms to other fields while continuing to enhance their performance.

TABLE DES MATIÈRES

REMERCIEMENTS II

RÉSUMÉ V

ABSTRACT VII

LISTE DES TABLEAUX XV

LISTE DES FIGURES XVII

LISTE DES SIGLES ET ABRÉVIATIONS XX

CHAPITRE 1 INTRODUCTION 1

1.1 Motivation pour les robots joueurs de tennis de table 1

1.2 Aperçu de la thèse 3

CHAPITRE 2 REVUE DE LITTÉRATURE 5

2.1 Généralités sur les robots joueurs de tennis de table 5

2.2 Architecture physique des systèmes robotiques 8

2.2.1 Architectures utilisées dans les systèmes robotiques joueurs de tennis de table 8

2.2.2 Une autre architecture à potentiel : les robots à câbles 11

2.3 Architecture générale de l'algorithme 12

2.4 Système de vision 13

2.4.1 Détection de la balle 13

2.4.2 Mesure de la position de la balle 16

2.5 Prédiction de trajectoire 18

2.4.3 Méthodes basées sur le modèle dynamique 19

2.4.4 Les méthodes basées sur les données 23

2.5 La stratégie de frappe 26

2.5.1 Le choix du point de frappe 26

2.5.2 La vitesse et l'orientation de la raquette 27
2.5.3 Cinématique inverse et choix de la configuration du robot 30
2.6 Génération de trajectoire 32
2.7 Contrôle 35
CHAPITRE 3 MOTIVATION SCIENTIFIQUE 39
3.1 Motivation 39
3.2 Problématique 39
3.2.1 Problème principal 39
3.2.2 Sous-problèmes 40
3.3 Objectif général 41
3.4 Objectifs spécifiques 41
CHAPITRE 4 ARTICLE 1 : REAL-TIME TRAJECTORY PREDICTION OF A PING-PONG BALL USING A GRU-TAE 42
4.1 Introduction 42
4.1.1 Model-based methods 43
4.1.2 Data-driven methods 44
4.2 Methodology 47
4.2.1 Implementation of the GRU-TAE 48
4.2.2 Experimentation setup 49
4.2.3 Trajectory dataset construction 49
4.2.4 Evaluation strategy 51
4.3 Results 53
4.3.1 Global accuracy 53
4.3.2 Simulation without noise 54
4.3.3 Simulation, with noise 55

4.3.4 Real dataset 56
4.4 Discussion 57
4.5 Conclusion 61
4.6 Statements and Declarations 62
4.7 Appendix 62
4.7.1 Implementation of the reference architectures 62
4.7.1.1 Recursive methods 62
4.7.1.2 TAE-based methods 63
4.7.1.3 Model-based architecture 63
4.7.2 Detailed results 64
4.8 References 69
CHAPITRE 5 ARTICLE 2 : BALL TRAJECTORY PREDICTION IN TABLE TENNIS BY COMBINING MACHINE LEARNING AND PHYSICAL MODELING 73
5.1 Introduction 74
5.1.1 Model-based methods 75
5.1.2 Data-driven methods 77
5.1.2.1 Neural network-based methods 77
5.1.3 Problem and objective 78
5.2 Methodology 79
5.2.1 Reference architectures 79
5.2.1.1 Data-driven methods 79
5.2.1.2 Model-based methods 79
5.2.2 Parameter identification of the differentiable model 81
5.2.3 Trajectory dataset construction 83
5.2.4 Evaluation strategy 83

5.3 Results 85
5.3.1 Parameters identification 85
5.3.2 Impact of dynamic parameters 86
5.3.3 Validation of the parameter's identification 87
5.3.4 Comparison of methods accuracy 88
5.4 Discussion 89
5.5 Conclusion 92
5.6 Statements and Declarations 93
5.7 Appendix 94
5.8 Références 95
CHAPITRE 6 ARTICLE 3 : DESIGN OF A MINIMAL MODEL-FREE CONTROL STRUCTURE FOR FAST TRAJECTORY TRACKING OF ROBOTIC ARM 99
6.1 Introduction 99
6.2 Materials and Methods 104
6.2.1 Equations of the Simulated Model 105
6.2.2 Implementation of the Control System 106
6.2.3 Test Procedure 106
6.3 Results 109
6.3.1 Results with the 3-DOF Simulated Model without Reduction Ratio 109
6.3.2 Results with the 3-DOF Simulated Model with a Reduction Ratio 109
6.3.3 Experimental Results 113
6.4 Discussion 116
6.5 Conclusions 121
6.6 Statements and Declarations 122
6.7 Appendix 122

6.7.1 A1. Estimation of the $\boldsymbol{Jm/(RMR)}$Ratio 122

6.7.2 A2. Case Example 1—Real 3-DOF Robotic Arm 123

6.7.3 A3. Case Example 2—Real 5-DOF Robotic Arm 123

6.8 References 123

CHAPITRE 7 ARTICLE 4 : REAL-TIME ALGORITHM FOR TABLE TENNIS WITH A DESKTOP ROBOTIC ARM 126

7.1 Introduction 127

7.2 Description of the solution 130

7.2.1 System architecture 130

7.2.2 Algorithm overview 131

7.2.3 Ball trajectory prediction 132

7.2.4 Robot planner 133

7.2.5 Motor control 137

7.2.6 Methodology 138

7.3 Results 139

7.3.1 Global performances with a collaborative user 139

7.3.2 Module Performances 140

7.3.2.1 Ball trajectory prediction 140

7.3.2.2 Robot planner 141

7.3.2.3 Robot control 142

7.4 Discussion 142

7.4.1 Global performance with a collaborative user 142

7.4.2 Module performances 144

7.4.2.1 Ball trajectory prediction 144

7.4.2.2 Robot planner 144

7.4.2.3 Robot control 145
7.5 Conclusion 146
7.6 References 147
CHAPITRE 8 DISCUSSION GÉNÉRALE 151
8.1 Développement d'une méthode de prédiction de trajectoire basée sur un modèle seq2seq récurrent 151
8.1.1 Prédiction de trajectoire basée sur un modèle seq2seq récurrent 151
8.1.2 Prédiction de trajectoire basée sur un modèle seq2seq récurrent 152
8.2 Structure de contrôle minimale 153
8.3 Algorithme pour le tennis de table 154
8.4 Contributions de recherche 155
8.5 Retour sur le projet 156
CHAPITRE 9 CONCLUSION ET RECOMMANDATIONS 159
RÉFÉRENCES 161

LISTE DES TABLEAUX

Tableau 2.1 Structure des robots existants pour le tennis de table tennis et leurs performances.... 6

Tableau 2.2 Architecture et positionnement des principaux bras robotiques utilisés dans les systèmes joueurs de tennis de table 11

Tableau 4.1 : List of time-series prediction methods used for comparison. 49

Tableau 4.2 : Comparison of training and computation time of tested methods 57

Tableau 5.1 : List of prediction methods used for comparison for ping-pong ball trajectory prediction 84

Tableau 5.2 : Results of parameters optimizations 85

Tableau 5.3 : Impact of the parameter accuracy on the trajectory prediction accuracy 86

Tableau 5.4 : Impact of the parameter optimization on real trajectory prediction accuracy 87

Tableau 5.5 Comparison of computation time for tested methods 88

Tableau 6.1 :Summary of tested parameters. 106

Tableau 6.2 : Computation of a command sequence. 108

Tableau 6.3 : Median relative RMSE according to the size of the NNs. 112

Tableau 6.4 : Comparison between the trajectory tracking accuracies (RMSE) of NNs for trajectory tracking on two real robot arms. 114

Tableau 6.5 : Guide for the design of a neural network-based control structure for position control, followed by an example of its application on the real 3- and 5-DOF systems. 120

Tableau 7.1 : Structure of existing robotic table tennis system, with their performance 127

Tableau 7.2 : Metrics used for evaluation of modules of the code 139

Tableau 7.3 : Summary of performances of our table tennis robotic arm during rallying 140

Tableau 7.4 : Performances of our table tennis robotic arm in non-rallying conditions, one shot at a time 140

Tableau 7.5 : Summary of the performances of the ball trajectory prediction module 141

Tableau 7.6 : Summary of the performances of the planner module 142

Tableau 8.1 : Contributions de recherche..... 156

LISTE DES FIGURES

Figure 1.1 Bras robotique de bureau développé par le laboratoire du professeur Maxime Raison. 3

Figure 2.1 : Robot Forpheus d'OMRON (image issue de [40]) 7

Figure 2.2 : IRB 1100 utilisé par Deepmind. (Image issue de [43]) 8

Figure 2.3 : Robots sériels utilisés pour le tennis de table (images issues de [11], [46]) 9

Figure 2.4 : Architecture générale de l'algorithme 12

Figure 2.5 : Méthodes de détection des objets mobiles 16

Figure 2.6 : Modèle idéal d'une caméra de stéréovision 17

Figure 2.7 : Méthodes de prédiction de la trajectoire de balle de tennis de table 18

Figure 2.8 : Schéma des paramètres et forces exercées sur la balle. 20

Figure 2.9 : Méthode d'estimation du spin à partir de son impact sur la trajectoire 21

Figure 2.10 : Architecture générale des méthodes récursives 24

Figure 2.11 : Architecture générale des « trajectory autoencoders » 25

Figure 2.12 : Utilisation d'un plan de frappe virtuel 27

Figure 2.13 : Paramètres de la frappe 29

Figure 2.14 : Stratégie de frappe avec trajectoire rectiligne 34

Figure 2.15 : Résumé des structures de contrôle existantes 38

Figure 4.1 : Performance of trajectory tracking with various NNs for (A) 3-DOF robotic arm, fast trajectories. 43

Figure 4.2 : Architecture of the GRU-TAE for ping-pong ball trajectory prediction. 48

Figure 4.3 : Comparison of RMSE for trajectory prediction for simulated and real data. 53

Figure 4.4 : RMSE comparison depending on the number of observations to predict for prefect simulated data, before rebound. 55

Figure 4.5 : RMSE comparison depending on the number of observations to predict for noisy simulated data, before rebound. 55

Figure 4.6 : Comparison of RMSE depending on the number of observations to predict for real data, before the rebound. 56

Figure 4.7 : Examples of prediction of a trajectory using different tested methods. 57

Figure 4.8 : General architecture of recursive methods for ping-pong ball trajectory prediction. 62

Figure 4.9 : General architecture of a TAE-based neural network for ping-pong ball trajectory prediction. 63

Figure 5.1 : Structure of existing methods for ping-pong ball trajectory prediction 75

Figure 6.1 : Model-free control structure for high-speed trajectory tracking. 102

Figure 6.2 : Model-free control structure for high-speed trajectory tracking. 103

Figure 6.3 : Design of the studied robotic arms for (A) simulated and real 3-DOF arms and (B) a real 5-DOF arm. 104

Figure 6.4 : Learning curve with both INN and GNN methods: (A) without reduction ratio, (B) with a reduction ratio R = 0.01. 110

Figure 6.5 : Relative performances of the NN-based control structure compared to the reference PD controller as a function of the $\boldsymbol{Jm/(RMR)}$ ratio. 111

Figure 6.6 : Relative performances of the NN-based control structure compared to the reference PD controller as a function of the accuracy of the PD controller and the step time of the direct dynamic integration. 112

Figure 6.7 : Relative performances of the neural networks to the reference PD controller as a function of the noise magnitude in the training data. 113

Figure 6.8 : Performance of trajectory tracking with various NNs for (A) 3-DOF robotic arm, fast trajectories (B) 3-DOF robotic arm, intermediate trajectories; (C) 5-DOF robotic arm, slow trajectories. 115

Figure 7.1 : A. Topology of the table tennis robotic arm. B. Real 3D-printed robot during play131

Figure 7.2 : Structure of the algorithm of the table tennis robot 132

Figure 7.3 : Conceptual image of the racket motion planning during a hit. 134

Figure 7.4 Model-free control structure of the robotic arm. 137

Figure 7.5 : Return parameters of the robotic system. 139

LISTE DES SIGLES ET ABRÉVIATIONS

La liste des sigles et abréviations présente, dans l'ordre alphabétique, les sigles et les abréviations utilisés dans le mémoire ou la thèse ainsi que leur signification. En voici quelques exemples :

CNN	Réseaux de neurones convolutionnels	CNN	Convolutional neural networks
DDL	Degrés de liberté	DOF	Degrees of freedom
GNN	Réseau de neurones global	GNN	Global Neural Network
GRU	Réseaux de neurones à « Unité récurrente fermée »	GRU	« Gated-Recurrent Unit » neural network
GRU-TAE	Autoencodeur à « Unité récurrente fermée»	GRU-TAE	Gated-Recurrent Unit Autoencoder
GRU-ED	Encodeur-Décodeur à « Unité récurrente fermée »	GRU-ED	Gated-Recurrent Unit Encoder-Decoder
IA	Intelligence artificielle	AI	Artificial intelligence
IL	Apprentissage par imitation		Imitation Learning
ILC	Contrôle par apprentissage itératif	ILC	Iterative learning control
iLQR	Régulateur Linéaire Quadratique Itératif	iLQR	Iterative Linear Quadratic Regulator
IMU	Centrale inertielle	IMU	Inertial Measurement Unit
INN	Réseaux de neurones indépendants	INN	Independant Neural Networks
ITTF	Fédération Internationale de Tennis de Table	ITTF	International Table Tennis Federation
LSTM	Réseaux de neurones à « Mémoire à long et court terme »	LSTM	« Long-Short Term Memory » neural network
MB	Méthode basée sur les modèles dynamiques	MB	Model-based method
MBS	Méthode basée sur les modèles dynamiques avec estimation du spin	MBS	Model-based method with spin estimation

NN	Réseau de neurones	NN	Neural Network
PD	Contrôleur Proportionnel-Dérivé	PD	Proportional–Derivative Controller
PID	Contrôleur Proportionnel-Intégral-Dérivé	PID	Proportional–Integral–Derivative Controller
RBF	Fonction de Base Radiale	RBF	Radial Basis Function
RCNN	Réseau de neurones convolutionnel récurrent	RCNN	Recurrent Convolutional Neural Network
RL	Apprentissage par renforcement	RL	Reinforcement learning
RMSE	Erreur quadratique moyenne	RMSE	Root-Mean-Squared-Error
RNN	Réseaux de neurones récurrents	RNN	Recurrent Neutral Network
RPM	Rotation Par Minute	RPM	Revolution Per Minute
seq2seq	Modèles « séquence à séquence »	seq2seq	« sequence to sequence » models
SO	Sous-Objectif	SO	Sub-Objective
TAE	Autoencodeur de Trajectoire	TAE	Trajectory Autoencoder
TVAE	Autoencodeur Variationnel de Trajectoire	TVAE	Trajectory Variational Autoencoder
VHP	Plan de frappe virtuel	VHP	Virtual Hitting Plane
YOLO	Algorithme “On ne regarde qu'une fois”	YOLO	algorithm “You Only Look Once”

CHAPITRE 1 INTRODUCTION

1.1 Motivation pour les robots joueurs de tennis de table

Le marché mondial de la robotique connaît une croissance rapide. Selon GlobalData, une entreprise leader dans le domaine des données et de l'analytique, ce marché pourrait tripler d'ici 2030, pour atteindre 218 milliards de dollars [1]. L'entreprise justifie ces prévisions par l'augmentation prévue du rythme d'installation de robots industriels dans les prochaines années. Toujours selon GlobalData, le parc mondial compterait déjà plus de trois millions de robots industriels. Cette forte progression s'explique par la multiplication des cas d'usage. L'intégration de systèmes robotiques avancés et de l'intelligence artificielle (IA) transforme de nombreux secteurs allant de l'industrie au grand public, en passant par la fabrication, la santé, l'automobile, le spatial, la logistique, l'agriculture, ou encore la sécurité et la défense [2].

Si les robots industriels sont très largement utilisés, les robots de services et les robots collaboratifs, ou cobots ouvrent de nouvelles opportunités. En robotique d'assistance, par exemple, ces robots sont principalement employés en milieu clinique, notamment dans les hôpitaux et les centres de réadaptation, mais leur usage à domicile se développe [3]. Dans [4], les principaux cobots existants sont recensés, en détaillant leurs capacités de charge, leur espace de travail et leur précision. Bien que les cobots soient moins rapides, moins précis, et disposent d'un espace de travail plus petit que les robots industriels [4], ils offrent des performances satisfaisantes pour de nombreuses applications. Cependant, leur coût encore élevé et leur manque d'adaptabilité constituent un frein majeur à leur adoption [3]. Ce constat effectué pour la robotique d'assistance peut s'appliquer à d'autres secteurs.

De leur côté, les robots personnels souffrent de performances encore plus limitées. De la même manière que les cobots se comparent aux robots industriels, les robots personnels se distinguent des cobots par une moindre rapidité, précision, capacité de charge et espace de travail, ce qui restreint leurs possibilités d'application. Dans [5], un état de l'art des robots de services est réalisé, incluant les robots « domestique et/ou personnels ». Ces travaux montrent que les robots domestiques sont principalement utilisés pour des applications d'aspirateur [6], [7], [8] ou de tonte de gazon.

L’intégration de l’intelligence artificielle et de l’apprentissage machine augmente néanmoins les capacités des cobots et robots personnels et ouvre ainsi de nouvelles opportunités pour les systèmes robotiques [4]. Selon Isabel Al-Dhahir, analyse principale chez GlobalData, « *Les progrès de l’IA permettent le développement de robots complexes à fonctionnalités multiples, alors que les robots n’étaient auparavant que des machines à fonction fixe, ce qui va multiplier leurs usages dans les années à venir* »[2]. En particulier, ces robots sont devenus plus intelligents et plus adaptables [5] pour des tâches sociales. Des robots domestiques ont ainsi été développés pour assister des personnes âgées [9] ou des robots compagnons.

Couplée aux nouvelles techniques d’impression 3D, l’IA devrait permettre de réduire l’écart de performance entre les systèmes robotiques personnels abordables et les cobots traditionnels, notamment en matière de dextérité, rapidité et précision. Le laboratoire dirigé par le professeur Maxime Raison s’est attaqué à la problématique du coût en développant un bras robotique de bureau personnel. Ce robot, à l’origine équipé d’un préhenseur adaptatif, dispose de cinq degrés de liberté. Il est portable et personnalisable, pour un coût en matériel inférieur à 3,000$ grâce à l’utilisation de l’impression 3D. Il est présenté à la Figure 1.1.

D’autre part, le tennis de table est un sujet de recherche actif en robotique depuis les années 1980 et le développement du premier robot joueur de tennis de table [10]. Ce sport constitue en effet un défi de taille pour les chercheurs en robotique en raison des exigences élevées en matière de dextérité, d’agilité, de rapidité et de précision nécessaires pour prédire et intercepter les trajectoires de la balle. Ces exigences en font un excellent banc d’essai, régulièrement utilisé par des laboratoires de recherche [11], des industries [12] et des entreprises telle que Google [13], afin de tester et démontrer les performances des nouvelles technologies.

Depuis les années 1980, de nombreux systèmes robotiques ont été conçus avec des capacités toujours plus importantes [11], [14], [15], [16], [17]. Si certains systèmes ont été développés pour couvrir la totalité du jeu du tennis de table, un nombre important d’études se sont focalisées sur des aspects spécifiques. Il s'agit par exemple de la prédiction des trajectoires [18], la détection de la pose de la raquette [19], la détection des effets dans la balle [20], [21] ou encore l’identification des stratégies humaines [22], [23], [24]. D’autres travaux ont été réalisés sur la génération de mouvements et les actions du robot, comme le renvoi de la balle [25], [26], [27], [28], [29], la visée

d’une zone spécifique de la table [30], [31], les échanges [32], [33], [34] et même le smash [17], [35].

L’objectif de cette thèse est de développer un système collaboratif de tennis de table, basé sur le prototype du laboratoire du professeur Maxime Raison afin de démontrer les performances atteignables par des bras robotiques de bureau, intégrant l’intelligence artificielle.

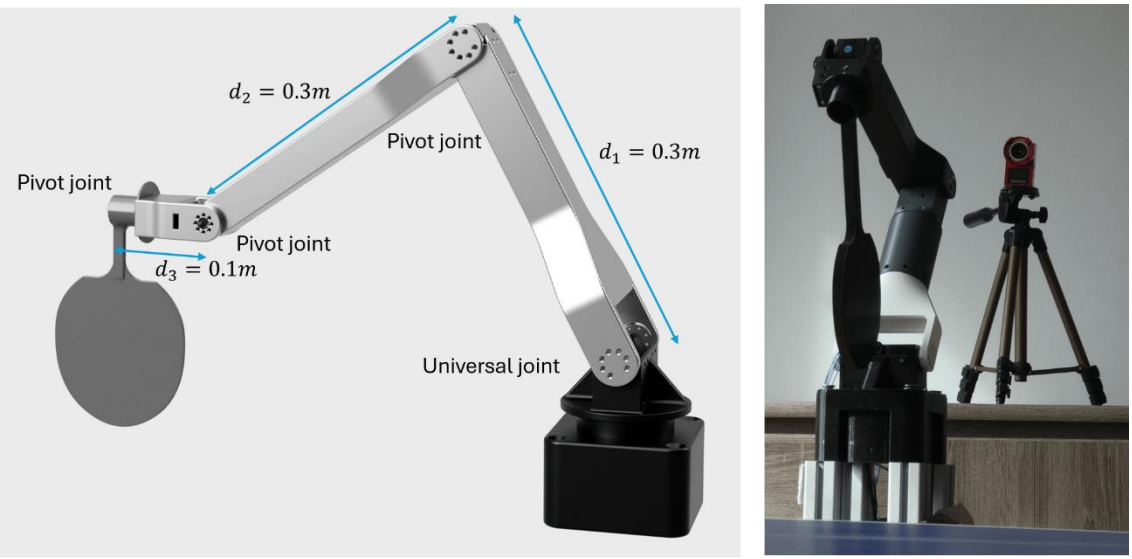


Figure 1.1 Bras robotique de bureau développé par le laboratoire du professeur Maxime Raison.

1.2 Aperçu de la thèse

Cette thèse s’inscrit dans l’objectif général de démontrer le potentiel des robots personnels, lorsqu’ils sont assistés par les techniques d’intelligence artificielle.

À la suite de ce chapitre introductif, le chapitre 2 présente le cadre théorique et une revue critique de littérature sur les systèmes robotiques joueurs de tennis de table. Cette analyse vise à fournir les bases nécessaires à une compréhension approfondie du projet. L’état de l’art conclut sur les défis et les limitations actuels de ces systèmes.

Le chapitre 3 expose la formulation du problème ainsi que les objectifs de recherche poursuivis dans cette thèse.

Les chapitres 4 et 5 présentent la prédiction de la trajectoire de la balle. Le chapitre 4 s’intéresse aux méthodes basées sur l’apprentissage machine. Une comparaison des différentes méthodes est

effectuée, permettant de mettre en valeur les avantages et inconvénients de chacune, puis une nouvelle méthode de prédiction est introduite combinant plusieurs architectures existantes. Le chapitre 5 s'intéresse quant à lui à l'intégration des méthodes d'apprentissage machine et des méthodes traditionnelles basées sur les modèles dynamiques.

Le chapitre 6 présente quant à lui ensuite une méthode pour le design d'une structure de contrôle pour le suivi de trajectoire rapide, sans connaissances préalables du modèle dynamique du système. Deux variations de cette méthode sont étudiées. Le chapitre conclut en présentant un guide pour l'implémentation optimale de cette structure de contrôle.

Le chapitre 7 présente l'algorithme développé pour le système collaboratif de tennis de table, basé sur le prototype du laboratoire du professeur Maxime Raison. Les méthodes utilisées dans les différentes parties de l'algorithme sont détaillées. Le chapitre conclut en présentant les résultats obtenus ainsi que les limitations du système avec des pistes d'amélioration.

Enfin, le chapitre 8 présente une discussion générale sur l'ensemble des travaux menés et les résultats obtenus au cours de cette thèse. Les limitations du système y sont développées, et des pistes pour de futurs travaux sont discutées.

CHAPITRE 2 REVUE DE LITTÉRATURE

Ce chapitre présente une revue de littérature sur les systèmes robotiques joueurs de tennis de table. Chaque section explore en détail un aspect spécifique du jeu, en présentant les différentes techniques développées jusqu'à présent, ainsi que leurs avantages et inconvénients. À la fin de chaque section, un encart présente les points clés que le lecteur pourra retenir.

La section 2.1 recense les principaux systèmes robotiques joueurs de tennis de table développés à ce jour et en présente les performances atteintes. Les architectures physiques et la structure générale de l'algorithme sont décrites dans les sections 2.2 et 2.3.

Un état de l'art est ensuite proposé pour les différents modules de ces algorithmes. La section 2.4 se concentre sur le système de vision, incluant le processus de détection de la balle et la détermination de sa position. La section 2.5 recense les méthodes existantes pour la prédiction de la trajectoire de la balle. Les stratégies de frappe adoptées par les robots sont détaillées dans la section 2.6. Finalement, la génération de trajectoire du robot et les structures de contrôles sont abordées respectivement dans les sections 2.7 et 2.8.

2.1 Généralités sur les robots joueurs de tennis de table

Le tennis de table est un sport qui trouve ses origines en Angleterre, à la fin du XIXe siècle. La table utilisée pour les compétitions est soumise aux normes de la Fédération Internationale de Tennis de Table (ITTF). Elle est rectangulaire, plane et horizontale et est située à une hauteur de 76 cm. La table mesure 274 cm de long et 152.5 cm de large et est séparée par un filet de 15.25cm de hauteur.

La balle de tennis de table est sphérique, de diamètre 40mm (norme en vigueur depuis le début des années 2000), et a une masse de 2.7g. Elle doit également répondre à des critères stricts de rigidité et de hauteur de rebond [36]. Lors de matchs avec des joueurs professionnels, elle peut atteindre une vitesse linéaire d'environ 100km/h et une vitesse de rotation d'environ 100tr/s [37].

En revanche, la forme, la dimension et le poids de la raquette ne sont pas réglementés [38]. Cependant, la palette de la raquette mesure environ 17cm de longueur et 15 cm de largeur et est constituée de différentes couches.

Depuis les années 1980 et le premier robot joueur de tennis de table [10], le tennis de table est un sujet de recherche actif en robotique. Il constitue en effet un défi pour les chercheurs en raison de

la dextérité et de l'agilité nécessaires, ainsi que de la rapidité et de la précision requises pour prédire la trajectoire de la balle. Ces exigences en font un excellent banc d'essai, régulièrement utilisé par des laboratoires de recherche [11], des industries [12] et des entreprises telle que Google [13], afin de tester et démontrer les performances des nouvelles technologies.

Depuis cette époque, de nombreux systèmes robotiques ont été désignés avec des capacités toujours plus avancées [11], [14], [15], [16], [17]. Si certains systèmes ont été développés pour la totalité du jeu du tennis de table, un nombre important d'études se sont focalisées sur des aspects spécifiques. Il s'agit par exemple de la prédiction de trajectoires [18], la détection de la pose de la raquette [19], la détection des effets dans la balle [20], [21] ou encore l'identification des stratégies humaines [22], [23], [24]. D'autres travaux ont été réalisés sur la génération de mouvements et les actions du robot, comme le renvoi de la balle [25], [26], [27], [28], [29], la visée d'une zone spécifique de la table [30], [31], les échanges [32], [33], [34] et même le smash [17], [35].

Plusieurs métriques permettent d'évaluer les performances des robots joueurs de tennis de table. Tout d'abord, le taux de renvoi du système à l'utilisateur est la mesure de la performance la plus utilisée [11], [25], [34]. Selon les capacités du système, le taux de balles touchées [31], ou l'erreur quadratique moyenne entre la position ciblée pour le rebond lors du retour et la position réelle du rebond [12] peuvent être mesurés pour préciser les résultats. Pour les systèmes compétitifs, le taux de victoire lors des matchs contre un utilisateur humain est utilisé [13], [39]. Enfin, il est essentiel de contextualiser ces résultats selon les conditions de jeu : contre un joueur compétitif [13], un joueur collaboratif [11], [12], [34], avec un autre robot [34], ou même en utilisant un lanceur de balles [31].

Le tableau 2.1 résume les principales architectures développées pour le tennis de table, ainsi que leurs performances.

Tableau 2.1 Structure des robots existants pour le tennis de table tennis et leurs performances

Modèle (entreprise, ville, pays)	Date de l'article	Résumé de l'architecture	Performance globale
IRB 1100 DeepMind (Google, Mountain View, USA)	2024	Bras sériel à 6 degrés de liberté (DDL) monté sur deux actionneurs linéaires Festo	Premier robot compétitif, "niveau d'un humain amateur lors de matchs compétitifs », avec 45% de victoires. [13]
Forpheus (Omron, Kyoto, Japan)	2019	Robot parallèle 5-axes. Robot Delta suspendu au-dessus de la	Réalise de longs échanges avec des joueurs humains, feedback fourni à l'utilisateur sur son jeu. Erreur

		table + 2 DDL pour la raquette	quadratique moyenne 22.5cm sur les retours. [12]
Agilus (KUKA, Tübingen, Germany)	2018	Robot sériel 6-DDL	87% de retours sur la table, sur 315 frappes [11]
WAM (Barrett, Tübingen, Germany)	2010-2013	Robot sériel 7-DDL	74.4% de retours. Monte à 88% après l'entraînement. [22]
PA10 (Mitsubishi, Nagoya, Japan)	2012	Robot sériel 7-DDL	70% de balles touchées, la balle est lancée avec une machine. [31]
Humanoïde (Zhejiang University, Hangzhou, China)	2012	30-DDL avec 7-DDL sur chaque bras	145 frappes avec un utilisateur collaboratif. [34]

Deux systèmes se démarquent aujourd'hui par leurs performances : le robot Forpheus développé par Omron (Kyoto, Japan) [12] et le système basé sur un robot IRB 1100 (ABB) développé par Google DeepMind (Mountain View, USA) [13].

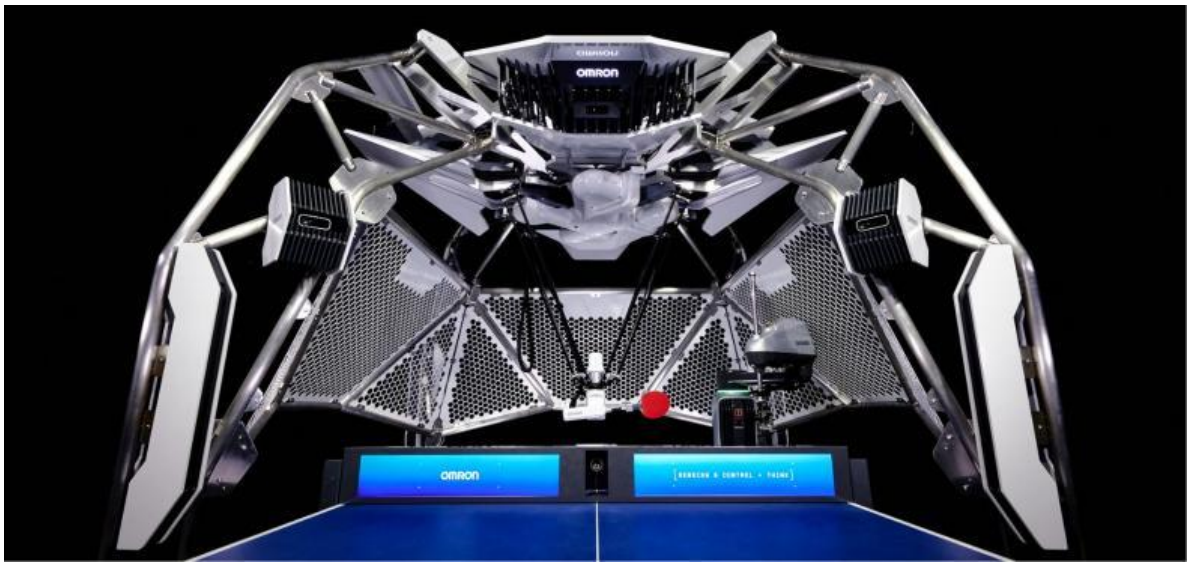


Figure 2.1 : Robot Forpheus d'OMRON (image issue de [40])

Le Robot Forpheus est capable de réaliser de longs échanges avec un utilisateur humain [12]. Ce robot est composé d'un bras parallèle Delta suspendu au-dessus de la table, sur lequel est montée la raquette. Cette architecture lui confère une grande réactivité, des accélérations importantes et un haut niveau de précision [41]. En particulier, Forpheus utilise les équations des modèles dynamiques pour renvoyer la balle avec précision à l'endroit ciblé. Il a démontré sa capacité à maintenir un échange de tennis de table avec des utilisateurs de bon niveau, imprimant différents effets dans la balle, tout en étant capable de produire un retour sur le jeu de l'utilisateur. Plusieurs vidéos de démonstration sont disponibles en ligne [42].

De son côté, le robot IRB 1100 de DeepMind est un système adaptatif capable de jouer des matchs compétitifs contre des joueurs humains de différents niveaux. Il s'agit du premier système robotique capable de jouer des matchs complets. Le robot est un bras sériel à 6 degrés de liberté (DDL), monté sur une base mobile sur un plan horizontal [39]. Contrairement à Forpheus, le système est basé sur l'apprentissage de polices de contrôle à partir de données provenant de simulations et d'expériences réelles [30], [32], ce qui élimine la nécessité d'utiliser des équations dynamiques dans le système final. Les résultats ont montré que l'IRB 1100 a « un niveau d'un joueur amateur de tennis de table, lors de matchs compétitifs », le robot ayant obtenu 45% de victoires lors de ses différents matchs [13].

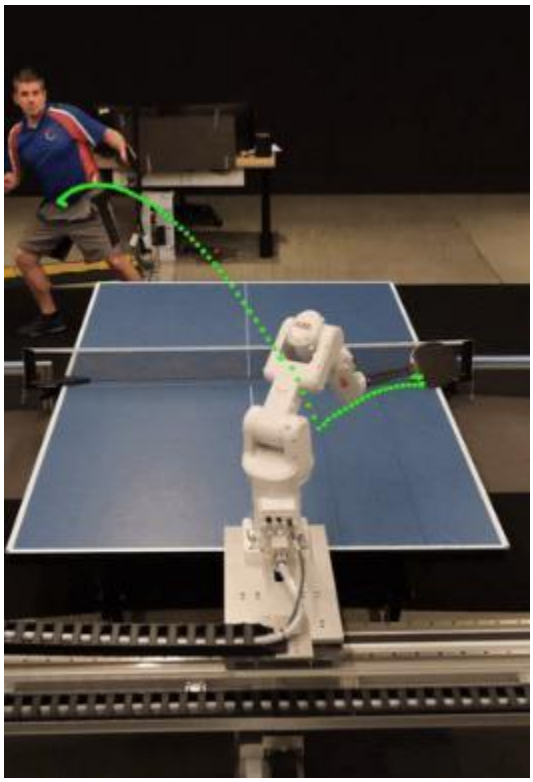


Figure 2.2 : IRB 1100 utilisé par Deepmind. (Image issue de [43])

2.2 Architecture physique des systèmes robotiques

2.2.1 Architectures utilisées dans les systèmes robotiques joueurs de tennis de table

La majorité des systèmes physiques incluent un bras robotique sériel, de fabrication industrielle, positionné derrière la table. Les bras sériels présentent plusieurs avantages : 1. un grand espace de

travail, car chaque articulation participe à l'étendre [44], [45] 2. Une cinématique simple, facile à calculer. 3. Une structure flexible et peu encombrante [2]. 4. Des algorithmes de contrôle bien établis et compatibles avec les robots industriels disponibles sur le marché [1].

Dans [11], la raquette est fixée sur un bras industriel à six DDL – KUKA Agilus KR6 R900 sixx – monté sur une base fixe. Un bras à sept DDL (Mitsubishi PA10-7C) est monté de façon similaire dans [31]. Dans [16], [25], un bras robotique à sept DDL – Barrett WAM – est suspendu au plafond, derrière la table. La principale limitation liée à ce positionnement et que l'espace de travail de ces systèmes ne couvre pas l'ensemble de la table, obligeant l'utilisateur à collaborer pour échanger des balles avec le robot.

Figure 2.3 : Robots sériels utilisés pour le tennis de table (images issues de [11], [46])

Pour résoudre ce problème, dans [47], un robot cartésien, constitué d'actionneurs linéaires est fixé directement sur la table. Les robots cartésiens facilitent les mouvements linéaires tout en offrant une haute précision, une bonne répétabilité et un grand espace de travail. Cependant, leur design est souvent encombrant et nécessite des installations complexes sur de grandes surfaces [48]. De plus, leur vitesse de mouvement est généralement limitée par leur structure mécanique, ce qui les rend moins adaptés aux applications à haute dynamique. Ils sont donc souvent mis de côté pour ce type d'applications, incluant le tennis de table.

Il est également possible de combiner un robot cartésien avec un robot sériel. Dans le système utilisé par DeepMind [13], [30], [39], le bras robotique sériel utilisé, un IRB 1100 à six DDL, est monté sur deux actionneurs linéaires. Ce montage permet au bras de se déplacer dans un espace de 8 mètres carrés et d'atteindre l'ensemble des balles.

Dans [34], un robot humanoïde est utilisé. Bien que ce système soit théoriquement capable de se déplacer et donc d'atteindre l'ensemble des balles, la démonstration n'exploite que le mouvement du bras du robot, qui possède sept DDL.

Les robots parallèles ont également été utilisés pour le tennis de table. Le robot Forpheus, développé par Omron, repose sur un robot Delta [41], qui possède trois DDL de translation, suspendu à une arche au-dessus de la table [12]. À l'extrémité du robot Delta, deux actionneurs supplémentaires permettent d'obtenir l'orientation souhaitée pour la raquette. Les robots parallèles, comme le Delta, se distinguent par leur rigidité élevée, leur grande précision et leur faible masse, ce qui les rend adaptés aux applications nécessitant une haute vitesse et une grande précision [45]. En revanche, leur structure rend la cinématique plus complexe et sujette à des singularités. Leur complexité mécanique les rend souvent plus chers et plus difficiles à designer et contrôler [45]. Dans le cas de Forpheus, son intérêt réside dans sa capacité à exécuter des mouvements rectilignes rapidement grâce à la structure Delta, lui permettant de frapper la balle peu après son rebond, ce qui améliore la précision globale du système. En revanche, Forpheus est un système robotique très imposant, rendant difficile une installation en dehors d'un laboratoire. Le fait qu'il soit placé directement au-dessus de la table exige également une réactivité élevée et une mise en place rapide pour intercepter les balles.

Enfin, une autre architecture notable est présentée dans [49]. Dans cette étude, un drone est utilisé pour renvoyer les balles de tennis de table, avec un taux de balles touchées de 40%.

Tout système robotique destiné à jouer au tennis de table doit bouger dans un espace tridimensionnel. Cependant, Anderson a affirmé que seuls cinq DDL étaient nécessaires pour manipuler la raquette de manière à renvoyer la balle de façon optimale [10]. Cette affirmation peut être vérifiée en notant que l'angle de rotation de la raquette dans le plan du plateau n'a aucune incidence sur le renvoi. Ainsi, l'architecture mécanique la plus simple permettant de jouer au tennis de table est un bras sériel à 5 DDL, tel que celui développé dans le laboratoire du professeur Maxime Raison et présenté à la Figure 1.1. Cette architecture peut également être utilisée pour des applications de la vie quotidienne – préhension, assistance pour les repas, jeux dans le cadre de la réadaptation – ou dans l'industrie.

Toutefois, comme nous l'avons vu, la plupart des systèmes développés sont des systèmes avec plus de cinq DDL, ce qui permet d'avoir des redondances et donc une flexibilité plus importante pour

optimiser d'autres paramètres lors de la frappe. Ces degrés de liberté sont utiles pour la réalisation de trajectoires rectilignes autour de la frappe [11], [12]. Ainsi, l'utilisation d'une architecture optimisée pour la réalisation de mouvements rectilignes ou la présence de DDL redondants est privilégiée.

Le tableau 2.2 résume les principaux robots utilisés pour le tennis de table.

Tableau 2.2 Architecture et positionnement des principaux bras robotiques utilisés dans les systèmes joueurs de tennis de table

Modèle (entreprise, ville, pays)	Date	Résumé de l'architecture	Positionnement
IRB 1100 DeepMind (Google, Mountain View, USA)	2024	Bras à 6 DDL ABB 1100 + 2 actionneurs linéaires Festo	Derrière la table, base du bras mobile sur une surface de 2m par 4m.
Forpheus (Omron, Kyoto, Japon)	2019	Robot parallèle 5-axes. Robot delta + 2 DDL pour la raquette	Suspendu à une arche au-dessus de la table.
Agilus (KUKA, Tübingen, Allemagne)	2018	Bras KUKA Agilus à 6 DDL	Derrière la table, base fixée.
WAM (Barrett, Tübingen, Allemagne)	2010-2013	Bras Barrett WAM à 7 DDL	Suspendu au plafond, derrière la table
PA10 (Mitsubishi, Nagoya, Japan)	2012	Bras 7-DDL	Derrière la table, base fixée.
Humanoïde (Zhejiang University, Hangzhou, China)	2012	Robot humanoïde, 30-DDL	Derrière la table

2.2.2 Une autre architecture à potentiel : les robots à câbles

D'autres architectures robotiques existent et pourraient être appliquées au tennis de table. Parmi celles-ci, les robots à câbles se distinguent par leur utilisation dans de nombreuses applications : la Skycam, utilisée dans les stades pour offrir des vues aériennes [50], les systèmes à retour haptiques [51], [52], [53], [54], ou la réadaptation [55], [56], [57], [58]. L'avantage des robots à câbles est leur structure légère, permettant des mouvements rapides avec dextérité [59], [60], [61], [62], dans des espaces de travail de taille importante [63], [64]. Ces architectures pourraient ainsi convenir aux exigences liées au tennis de table.

Cependant, plusieurs défis freinent actuellement leur développement. En particulier, la nécessité de maintenir une tension constante dans les câbles pour éviter le jeu et les vibrations complexifie

le contrôle de ces systèmes. Il est également nécessaire d'effectuer une maintenance régulière des robots, notamment pour vérifier que les câbles ne se détendent pas avec le temps [65].

Résumé :

- Architectures variées déjà appliquées au tennis de table (sériel, parallèle, cartésien, hybride)
- Systèmes physiques placés derrière la table, ou au-dessus lorsque c'est possible.
- Utilisation de cobots avec des composants industriels.
- Minimum de 5 DDL nécessaire, mais DDL redondants dans l'essentiel des systèmes, permet d'optimiser d'autres paramètres pendant la frappe.

Gap : Aucun robot personnel ou robot de bureau n'a été utilisé pour le tennis de table.

2.3 Architecture générale de l'algorithme

L'architecture générale de l'algorithme repose sur sept étapes principales présentées à la Figure 2.4. Tout d'abord, la détection de la balle et la prédiction de sa trajectoire. Dans un second temps, l'identification des points de frappes possibles ainsi que des vitesses et orientations cibles associées pour la raquette. Ensuite, la cinématique inverse est calculée et le point de frappe optimal est sélectionné. Enfin, la trajectoire du robot est générée, et la structure de contrôle s'assure du suivi

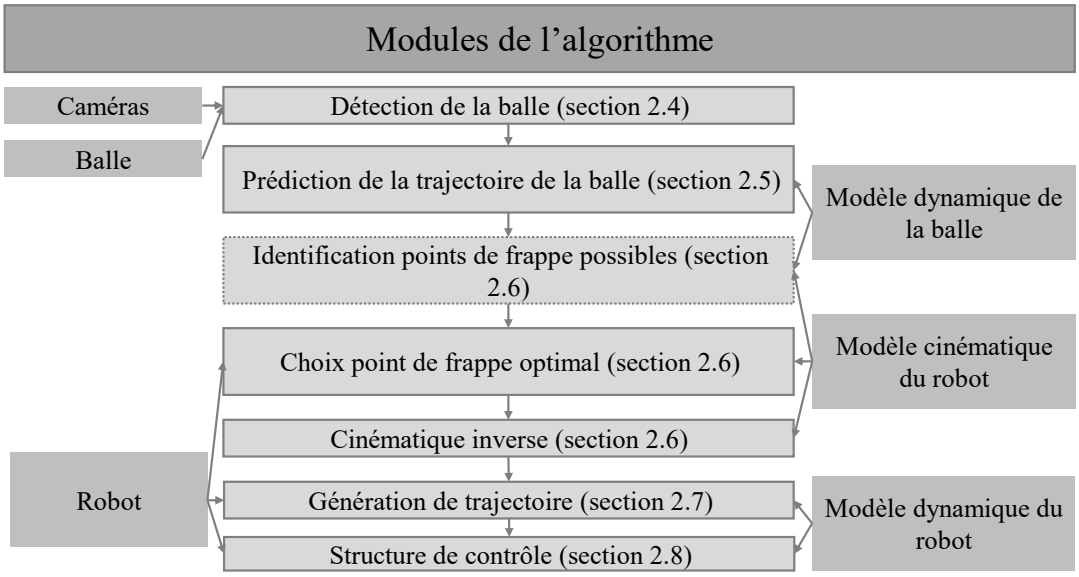


Figure 2.4 : Architecture générale de l'algorithme

de trajectoire. Ces modules seront détaillés dans les sections suivantes.

Dans un premier temps, la détection de la balle s'effectue en observant et mesurant ses positions successives à l'aide de caméras.

À partir de ces positions mesurées, la trajectoire de la balle est prédite en temps-réel par un module de simulation utilisant le modèle dynamique de la balle et d'autres techniques d'apprentissage.

Une fois la trajectoire de la balle prédite, le planificateur identifie les points de frappe possibles, ainsi que les vitesses et orientations cibles correspondantes pour que la raquette renvoie la balle à un point précis et en un temps donné.

Le point de frappe optimal est ensuite déterminé en tenant compte des contraintes physiques et de l'architecture du système robotique.

Le module de cinématique inverse calcule les positions et les vitesses articulaires nécessaires au moment de la frappe pour l'ensemble des configurations possibles.

Finalement, les trajectoires articulaires sont générées et le module de contrôle s'assure du suivi de trajectoire du bras robotique.

Selon l'architecture du système, l'ordre de certaines étapes peut varier, et certaines d'entre elles peuvent être effectuées simultanément. Cependant, ces sept modules constituent la base de l'algorithme de tous les systèmes.

2.4 Système de vision

La première étape de l'algorithme utilisé dans les systèmes robotiques capables de jouer au tennis de table est la détection de la balle et la mesure de sa position dans l'environnement. Pour cela, différentes méthodes peuvent être utilisées.

2.4.1 Détection de la balle

La détection et le suivi d'un objet en mouvement à partir de séquences vidéo constituent un sujet de recherche à part entière, utilisé dans de nombreuses applications.

La détection d'objets mobiles repose sur des algorithmes de traitement d'images visant à extraire des objets en mouvement à partir d'une séquence d'images. Elle s'appuie sur la détection de points d'intérêt – ou *features* –, tels que les contours, les couleurs ou les textures [66]. Plusieurs revues de littérature ont été consacrées aux méthodes de détection et de suivi de balles dans le sport [67], [68]

D'après [69], les trois principales approches les plus populaires pour la détection sont les méthodes de flux optique [70], la soustraction d'arrière-plan [71], [72] et la soustraction d'images [73].

La méthode par flux optique consiste à estimer le déplacement d'un champ entre deux images successives en identifiant des points de correspondances entre elles [74]. Cependant, cette technique requiert une quantité de calcul importante, ce qui la rend plus lente que les autres approches et difficilement applicable en temps-réel, notamment dans des sports rapides comme le tennis de table.

La soustraction de l'arrière-plan, quant à elle, est largement utilisée pour sa grande simplicité. Bien qu'elle soit facile à implémenter, cette méthode est sensible aux changements de luminosité [75] et doit donc être appliquée de préférence dans un environnement contrôlé.

Enfin, la soustraction d'images est une technique fondamentale en vision par ordinateur, avec une faible charge de calcul. En revanche, cette méthode est très sensible au bruit présent dans les images [76], ce qui peut limiter sa précision dans certaines conditions [69].

Des approches combinant soustraction d'arrière-plan et soustraction d'images ont été développées, permettant de réduire la sensibilité aux changements d'éclairage et au bruit [77], [78], [79].

Les techniques d'apprentissage machine basées sur les réseaux de neurones convolutionnels (CNN) se distinguent par leur large spectre d'applications [80]. De nombreux modèles basés sur les CNN ont été développés pour la détection et le suivi dans plusieurs sports : football [81], [82], cricket [83], [84], tennis [85] ou encore hockey [86].

Selon [68], deux grandes catégories de détecteurs basés sur les CNN existent. Premièrement, ceux opérant en une seule étape tel que les algorithmes « You Only Look Once » (YOLO) [87], [88], les « single shot detection » [89], les « RETINANET » [90], les « SQUEEZEDET » [91] ou encore les « EfficientDET » [92]. Deuxièmement, ceux opérant en deux étapes : Region-CNN [93], Fast RCNN [94], Faster-RCNN [95], Feature Pyramid Network [96], Mask-RCNN [97] ou encore YOLOv4.

En revanche, ces techniques nécessitent une grande quantité de données labellisées pour l'apprentissage. Pour pallier à ce manque, une base de données labellisées pour le tennis de table est présentée dans [98]. Il consiste en cinq vidéos de 10 à 25 minutes pour l'entraînement, et sept courtes vidéos pour la phase de test. Malgré ces initiatives, la collecte de données labellisées spécifiques reste un défi.

Ces techniques de détection ont été appliquées sur les systèmes robotiques utilisés pour le tennis de table. Le robot développé par Google DeepMind [30] utilise deux caméras opérant à 125 Hz, avec un modèle de détection 2D entraîné sur environ deux heures de vidéos durant lesquelles la position de la balle a été traquée en 3D. Dans [99], un algorithme de détection est basé sur un modèle convolutionnel d'une seule cellule, appliqué à des images filtrées. Ce modèle, basé principalement sur la reconnaissance de la couleur de la balle, atteint une erreur moyenne de 19,9 mm.

Le robot Forpheus développé par Omron est quant à lui basé sur trois caméras opérant en stéréo vision à 80 Hz. Les détails de l'algorithme de détection ne sont cependant pas précisés.

Dans [11], le système de vision est basé sur un ensemble de caméras opérant à 150 Hz. Une méthode de soustraction d'images est utilisée pour minimiser le temps de calcul. Après avoir réalisé la soustraction, l'image est convertie en image binaire avec un seuil. Chaque zone blanche est évaluée – ratio, aire, périmètre, distance de la détection précédente notamment – et le centre de la zone la plus probable est ensuite traité comme la position de la balle dans l'image. La précision de cette méthode a été mesurée, avec une erreur moyenne sur la position de 6.6mm. D'autres méthodes similaires ont été implémentées avec une précision comparable dans [100], avec une précision de 9mm.

Dans [101], deux caméras fonctionnent à 125 Hz pour détecter la balle. La balle est détectée en identifiant les objets ronds, après soustraction de l'arrière-plan, avec une précision de 5 mm en moyenne. Une approche similaire est décrite dans [102], associée à un seuil pour obtenir une image binaire. Une source lumineuse constante est ajoutée pour stabiliser l'éclairage.

La précision de caméras infrarouges pour la détection de la balle a également été étudiée [11]. Pour cela, des caméras de la marque OptiTrack ont été utilisées, et la balle a été recouverte d'un adhésif réfléchissant la lumière infrarouge. Selon l'entreprise, la précision de ces caméras est inférieure au millimètre. L'un des principaux avantages des caméras infrarouges est l'absence de traitement d'image en couleur, ce qui réduit les temps de calcul. Les caméras infrarouges sont également moins sensibles au bruit et aux conditions lumineuses, ce qui permet une utilisation dans des environnements non contrôlés. Cependant, leur principal inconvénient réside dans la nécessité de modifier la balle en y ajoutant un ruban adhésif réfléchissant. Cette modification peut altérer la

dynamique de la balle, compliquant ainsi l'utilisation de modèles dynamiques pour l'estimation de la trajectoire [103].

S'il n'existe pas à notre connaissance de travaux ayant défini la fréquence optimale pour les caméras, il existe un consensus autour de 100Hz. La majorité des systèmes de vision opère à une fréquence comprise entre 80 [12] et 150 Hz [11]. Une telle fréquence permet de maintenir un temps de réaction du système faible, tout en préservant une quantité de calculs acceptable.

La Figure 2.5 résume les différentes méthodes pour la détection d'objets mobiles.

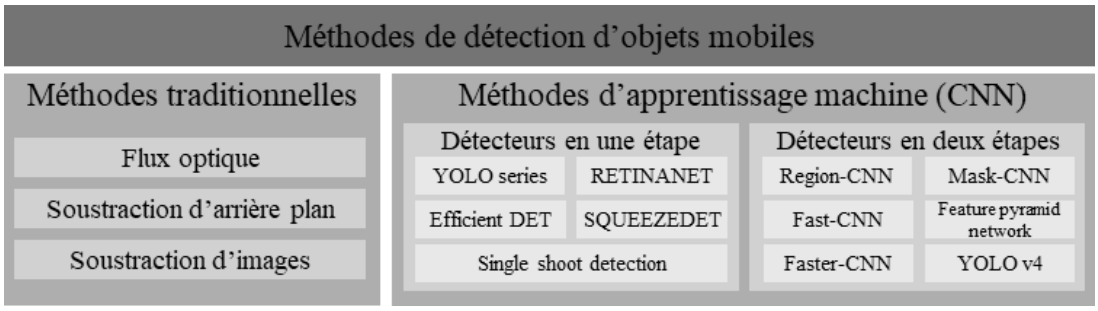


Figure 2.5 : Méthodes de détection des objets mobiles

2.4.2 Mesure de la position de la balle

La méthode la plus simple et la plus répandue pour mesurer la position d'un objet dans un espace en trois dimensions est la stéréovision [104]. Cette technique repose sur l'utilisation d'au moins deux caméras pour reconstruire la position d'un objet dans un environnement à trois dimensions à partir de plusieurs images d'une même scène prise de points de vue différents.

Dans le cas le plus simple, où les plans images des caméras sont coplanaires, la position de l'objet peut être calculée de la manière suivante :

Soient x_l et y_l les coordonnées du point d'intérêt sur l'image capturée par la caméra gauche, soient x_r et y_r les coordonnées correspondantes sur l'image de la caméra droite. Les coordonnées (x, y, z) de l'objet dans le repère de la caméra gauche peuvent alors être déterminées en appliquant les équations suivantes :

$$\frac{z}{f} = \frac{x}{x_l} = \frac{x-b}{x_r} = \frac{y}{y_l} = \frac{y}{y_r} \tag{1}$$

d'où

$$x = \frac{x_l z}{f} \tag{2}$$

$$y = \frac{y_l z}{f} \tag{3}$$

$$z = \frac{fb}{(x_l - x_r)} \tag{4}$$

avec f la distance focale de la caméra. Cependant à moins de disposer d'une caméra stéréo, le modèle de vision n'est pas idéal et il faut passer par une étape de calibration. Cette étape permet aussi de trouver d'autres paramètres des caméras tels que leur distorsion ou leur distance focale. La Figure 2.6 illustre les différents paramètres.

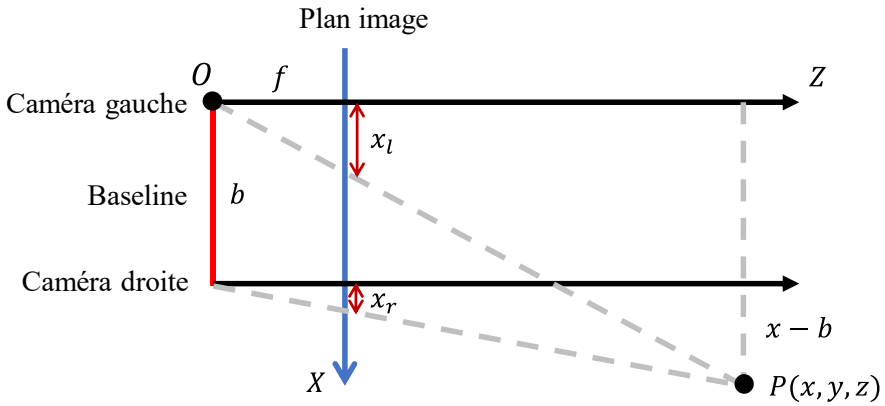


Figure 2.6 : Modèle idéal d'une caméra de stéréovision

Il est possible d'utiliser plus de deux caméras afin d'obtenir une redondance des informations et ainsi améliorer la précision des mesures de position. Dans ce cas, il devient nécessaire de résoudre un problème de minimisation. La méthode des moindres carrés est généralement employée, combinée à un algorithme de filtrage pour éliminer les mauvaises détections, ou *outliers*, comme l'algorithme RANSAC [105] ou un filtre de Kalman [11].

La stéréovision est utilisée dans la quasi-totalité des systèmes robotiques pour le tennis de table [11], [12], [30], [99], [101], la plupart des dispositifs adoptant une approche redondante en augmentant le nombre de caméras.

Une exception notable est le travail présenté dans la référence [102]. Ici, le système de vision ne comporte qu'une seule caméra et une source lumineuse fixées au plafond. La position de la balle est déterminée à partir de celle de son ombre sur la table, après une phase de calibration préalable.

Résumé :

Détection et suivi d'objets en mouvement :

- Méthodes principales : flux optique, soustraction d'arrière-plan, soustraction d'images
- Caméras infrarouges (+) Facile, robustes, calculs minimaux (-) modification de la balle nécessaire
- Apprentissage machine (CNN) : Performances élevées, mais nécessitant de grandes quantités de données labellisées pour l'entraînement.

Calcul position

- Stéréovision – Minimum de deux caméras, redondance possible.
- Étape de filtrage pour supprimer les mauvaises détections.

2.5 Prédiction de trajectoire

La prédiction en temps-réel de la trajectoire d'objets mobiles, et en particulier de celle d'une balle de tennis de table, basée sur des observations visuelles est un challenge pour les systèmes robotiques [106]. En effet, les positions successives calculées à partir du système de vision sont souvent bruitées et peuvent contenir des « outliers » ou des observations manquantes [107]. Le temps de calcul est également un facteur critique, parce qu'il doit être significativement plus rapide que la balle elle-même [108]. De façon plus large, la robotique à haute dynamique requiert souvent de prédire l'évolution de quantités physiques en temps-réel à partir d'observations précédentes.

Les méthodes de prédiction de la trajectoire de la balle de tennis de table ont été largement étudiées. Elles se résument principalement en deux catégories – les méthodes traditionnelles basées sur les

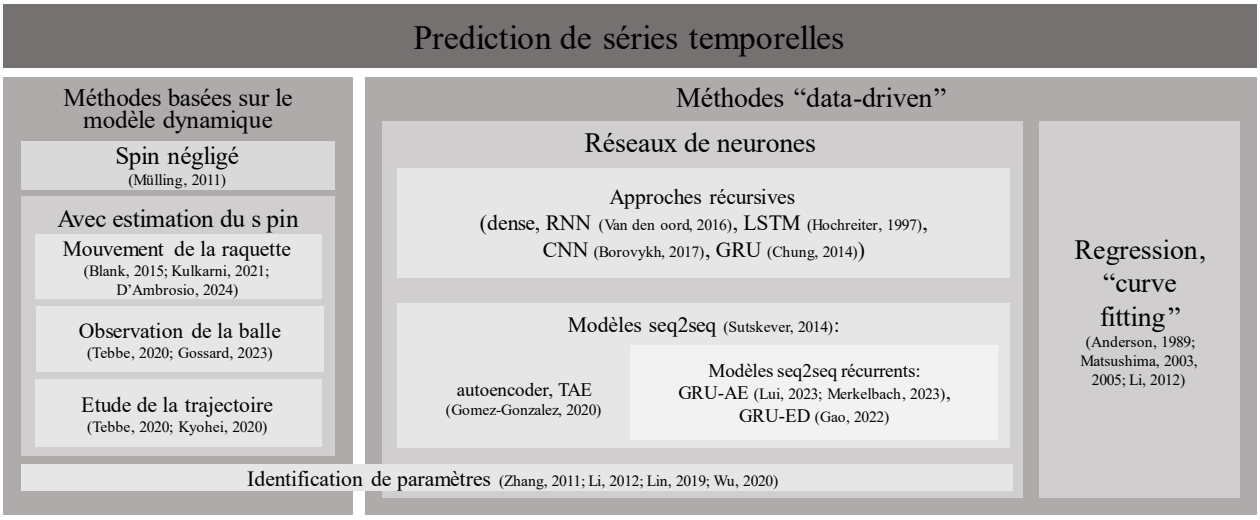


Figure 2.7 : Méthodes de prédiction de la trajectoire de balle de tennis de table

modèles dynamiques et les méthodes par apprentissage - qui seront détaillées dans les prochains paragraphes. La Figure 2.7 résume ces méthodes.

2.4.3 Méthodes basées sur le modèle dynamique

Les méthodes basées sur le modèle dynamique sont souvent privilégiées pour modéliser et prédire les trajectoires des systèmes robotiques à haute dynamique [109], [110]. Ces modèles, généralement basés sur des systèmes d'équations différentielles, présentent l'avantage d'être rapides à calculer, bien documentés, et représentatifs de la réalité physique des objets en mouvement. Malgré tout, une limitation majeure réside dans la difficulté d'estimer certains paramètres essentiels, même lorsque les modèles sont bien étudiés [111]. Pour le tennis de table, l'estimation du spin est un bon exemple de paramètre essentiel à prendre en compte. Ceci s'explique par le fait que le spin influe significativement sur la trajectoire de la balle, notamment lors du rebond avec la table [106]. Cependant, il est très difficile de calculer le spin à partir d'images provenant de caméras standards [12].

Face à cette difficulté, certains travaux choisissent de ne pas considérer le spin dans la prédiction de trajectoire [110].

Dans le cas, le modèle dynamique de la balle est le suivant :

$$m\ddot{\boldsymbol{x}}(\boldsymbol{t}) = -m\boldsymbol{g} - m\ddot{\boldsymbol{x}}_{\boldsymbol{D}}(t) = -m\boldsymbol{g} - C_D\|\dot{\boldsymbol{x}}(t)\|\dot{\boldsymbol{x}}(t) \qquad (\mathbf{5})$$

Avec $x(t), \dot{x}(t), \ddot{x}(t)$ correspondant respectivement à la position [m], la vitesse [m/s] et l'accélération [m/s²] de la balle à l'instant t, dans le référentiel global ; m sa masse, $m\ddot{x}_D$ la force aérodynamique et C_D la constante liée à la force aérodynamique, tel que $C_D = \frac{-1}{2} k_D \rho_a A m^{-1}$, avec $k_D = 0.4$, le coefficient aérodynamique, $\rho_a = 1.29 kg/m^3$ la densité de l'air et $A = r^2\pi$ l'aire de la section transversale de la balle [11].

Cependant, négliger le spin entraîne une baisse significative des performances. Pour compenser cette limitation, les systèmes choisissent souvent de frapper la balle juste après le rebond [12], ou demandent à l'utilisateur de limiter les effets dans la balle.

Lorsque la vitesse de rotation de la balle est prise en compte, il est nécessaire d'ajouter la force de Magnus, qui représente la force exercée sur la balle par sa vitesse de rotation. L'équation (1) devient alors :

$$m\ddot{\boldsymbol{x}}(\boldsymbol{t}) = -m\boldsymbol{g} - m\ddot{\boldsymbol{x}}_{\boldsymbol{D}}(t) + m\ddot{\boldsymbol{x}}_{\boldsymbol{M}}(t) = -m\boldsymbol{g} - C_D\|\dot{\boldsymbol{x}}(t)\|\dot{\boldsymbol{x}}(t) + C_M\{\dot{\boldsymbol{x}}(t) \times \boldsymbol{\omega}(\boldsymbol{t})\} \quad (\mathbf{6})$$

Avec ω correspondant au spin de la balle, $m\ddot{x}_M$ à la force de Magnus et C_M à la constante de Magnus [11], tel que $C_M = \frac{1}{2}k_m\rho_a Arm$, avec $k_m = 0.6$ le coefficient de lift. La figure 2.8 illustre les différents paramètres.

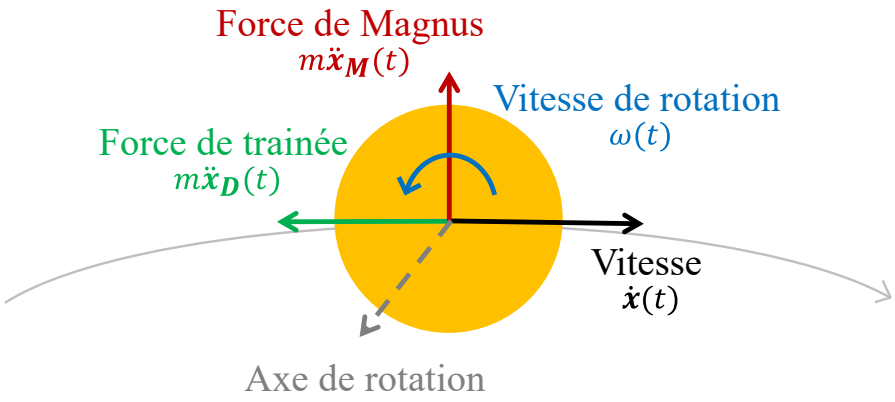


Figure 2.8 : Schéma des paramètres et forces exercées sur la balle.

Pour inclure le spin dans les prédictions à l'aide de modèles dynamiques, différentes méthodes pouvant être réparties en trois approches principales ont été explorées. À partir du mouvement de la raquette pendant la frappe, à partir de l'observation de la balle [107], ou à partir de l'étude de sa trajectoire [12].

Dans [112], une classification des frappes est réalisée en analysant le mouvement du joueur. Sur la vidéo présentant le robot Forpheus d'Omron, des marqueurs sont visibles sur la raquette et sont utilisés pour la classification des frappes [42]. Cette classification peut également être effectuée avec des centrales inertielles (IMU) [20], [113]. Cependant, seule une classification est réalisée, et ces travaux ne fournissent pas de données quantitatives sur la vitesse de rotation.

Certaines études ont été réalisées pour calculer la vitesse de rotation à partir d'observations de la balle avec des caméras ultrarapides [106], [107]. La vitesse de rotation peut alors être obtenue en traquant le logo [100] ou un pattern dessiné sur la balle [107]. Néanmoins, ces méthodes requièrent un système de vision souvent coûteux parce que les caméras doivent avoir une résolution et une fréquence élevées. De plus, une quantité importante de calcul est nécessaire, car il faut traiter les images en amont pour détecter le logo ou le pattern. Dans [107], le spin est ainsi estimé en 60ms à partir de 10 images successives des caméras.

Enfin, la vitesse de rotation de la balle peut être estimée à partir de l'observation de sa trajectoire. Certaines études estiment le spin à partir de l'effet de la force de Magnus – force causée par la rotation de la balle [106].

Pour cela, il est nécessaire de connaître les autres paramètres de la balle, et notamment sa vitesse linéaire. Comme une légère erreur sur la vitesse linéaire a un impact majeur sur l'estimation de la force de Magnus, une étape de filtrage des *outliers* est souvent ajoutée [106]. Une méthode similaire est détaillée dans [12]. Dans un premier temps, la position initiale et la vitesse initiale sont calculées. La position initiale correspond à la première position de la balle détectée. La vitesse linéaire est quant à elle obtenue de façon itérative, en utilisant la première et la dernière position mesurée (ou la dernière avant le rebond). Cette méthode de calcul permet de réduire l'impact des outliers et de s'affranchir de l'étape de filtrage, coûteuse. Une fois ces deux paramètres obtenus, au lieu d'estimer la vitesse de rotation via la force de Magnus, le spin est obtenu à partir de son impact sur la trajectoire. Pour cela, la fonction de coût suivante est minimisée :

$$L(\omega) = \sum_{i=1}^{N} e_i(\omega) \tag{7}$$

Où e_i est l'erreur entre la position prédite et la position mesurée, au pas de temps i.

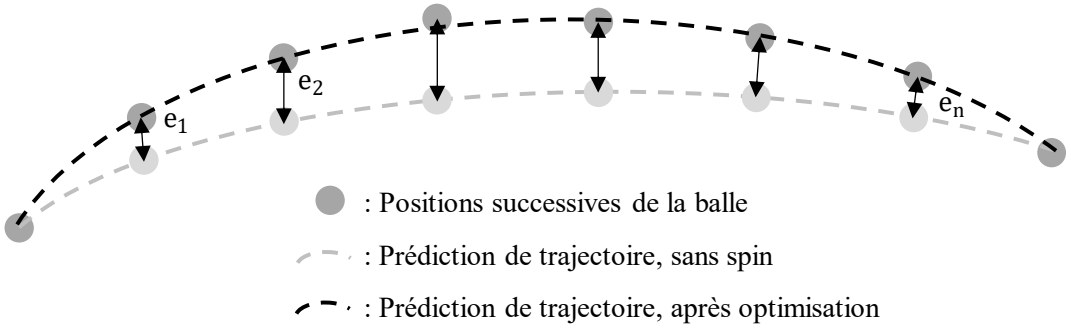


Figure 2.9 : Méthode d'estimation du spin à partir de son impact sur la trajectoire

Le spin n'est pas le seul paramètre influençant la précision de la prédiction de la trajectoire. Il est également crucial d'identifier les autres paramètres du modèle dynamique. Plusieurs d'entre eux sont soumis aux normes de la Fédération Internationale de Tennis de Table (ITTF), comme le diamètre (40mm), la masse (2.7g), la rigidité, la hauteur du rebond ou encore la sphéricité [38]. Cependant, il est important de noter que des tolérances existent pour ces paramètres, comme pour

la masse – entre 2.67g et 2.77g – ou le coefficient de restitution lors du rebond – entre 0.89 et 0.92 pour une chute libre de 0.3m sur un bloc d'acier. Il a également été démontré que ce coefficient de restitution diminue avec l'augmentation de la vitesse d'impact [114], [115]. Chaque marque de balle présente des propriétés spécifiques, influencées notamment par les matériaux utilisés. Finalement, lorsque le système de caméras utilise le rayonnement infrarouge, l'adhésif ajouté sur la balle modifie fortement la majorité de ces paramètres.

Ainsi, il est souvent nécessaire d'estimer expérimentalement certains paramètres dynamiques. Dans [116], le coefficient de restitution de la table lors du rebond est obtenu en mesurant les hauteurs maximums avant et après le rebond sur un ensemble de 25 données. Dans [111], la modélisation du rebond a été réalisée par analyse d'images. Dans [3], le modèle pour le rebond est donné, dans le cas d'un roulement sans glissement, c'est-à-dire avec une vitesse faible. Dans cette étude, les paramètres dynamiques ont été obtenus expérimentalement avec une balle en ABS – l'auteur précise que les paramètres peuvent différer avec une balle en celluloïd. Le modèle obtenu pour le rebond est le suivant :

$$\dot{\boldsymbol{x}}' = \boldsymbol{A_v}\dot{\boldsymbol{x}} + \boldsymbol{B_v}\boldsymbol{\omega} \tag{8}$$

$$\boldsymbol{\omega}' = \boldsymbol{A_w}\dot{\boldsymbol{x}} + \boldsymbol{B_w}\boldsymbol{\omega} \tag{9}$$

Avec $\dot{x}$ et $\dot{x}'$ les vitesses avant et après la frappe, respectivement, et ω et ω' le spin avant et après la frappe, respectivement. De plus :

$$\boldsymbol{A_v} = \begin{bmatrix} \frac{3}{5} & 0 & 0 \\ 0 & \frac{3}{5} & 0 \\ 0 & 0 & -C_{rz} \end{bmatrix}, \boldsymbol{B_v} = \begin{bmatrix} 0 & -\frac{2r}{5} & 0 \\ \frac{2r}{5} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \boldsymbol{A_w} = \begin{bmatrix} 0 & \frac{3}{5r} & 0 \\ \frac{-3}{5r} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \boldsymbol{B_w} = \begin{bmatrix} 2/5 & 0 & 0 \\ 0 & 2/5 & 0 \\ 0 & 0 & C_{\omega z} \end{bmatrix}$$

$$C_{\omega z} = 1 - \frac{3\tau c b}{2r^2}\frac{\|\vec{v}^P\|}{\|\vec{v}^M\|}|\dot{x}_z|^{0.75}u \tag{10}$$

Où $u = \mu(1 + C_{rz})\frac{|\dot{x}_z|}{\|\vec{v}^P\|}$, $\|\vec{v}^M\| = \sqrt{(\omega_z\tau)^2 + a^2}$. Avec $\mu = 0.22$ le coefficient de frottement dynamique entre la balle et la table, $C_{rz} = 0.9$ le coefficient adimensionnel de restitution de la table selon l'axe z, $\tau = 0.5ms$ la durée du contact table/balle, $r = 0.02m$ le rayon de la balle et $\vec{\boldsymbol{v}}^{\boldsymbol{P}}$ la vitesse tangentielle entre la balle et la table définie par $\vec{v}^P = [\dot{x}_x - r\omega_y\ ;\ \dot{x}_y + r\omega_x\ ; 0]$.

Cependant, ces estimations requièrent des expérimentations coûteuses en temps. Dans la pratique, les paramètres sont souvent tirés de valeurs moyennes issues de la littérature.

2.4.4 Les méthodes basées sur les données

Les méthodes basées sur les données (data-driven) ne nécessitent pas de modèle dynamique, et les prédictions de la trajectoire de la balle sont réalisées uniquement à partir des observations.

Plusieurs études ont réalisé la prédiction de trajectoire en mappant les observations passées avec des techniques de régression, notamment des méthodes polynomiales ou des approximations linéaires [33], [47], [117]. Cependant, en raison de la nature hautement non linéaire de la dynamique de la balle, ces approches génèrent des erreurs importantes [108].

Récemment, les recherches sur les méthodes basées sur les données et l'apprentissage machine ont progressé de façon importante et ces méthodes ont été largement appliquées à la prédiction de séries temporelles, et notamment la prédiction des trajectoires de balle de tennis de table.

Trois approches principales ont été utilisées : l'identification de paramètres, l'utilisation de modèles récursifs et les modèles sequence-to-sequence (seq2seq). Ces approches sont détaillées dans les sections suivantes.

Premièrement, l'identification de paramètres consiste à apprendre certains paramètres spécifiques de la trajectoire. Dans [118], deux réseaux de neurones sont utilisés pour estimer les coefficients des polynômes de degré 2 approximant la trajectoire de la balle avant et après le rebond. Un autre réseau de neurones a également été entraîné pour déterminer le point de frappe entre la balle et la raquette à partir de seulement quatre observations de la balle [102]. Cependant, ces méthodes manquent de précision, car certains paramètres demeurent inconnus.

Ensuite, les approches récursives prédisent directement les positions successives de la balle. La méthode générale consiste à utiliser un réseau de neurones pour prédire la position suivante en fonction des observations précédente, de manière itérative. L'algorithme peut être adapté en changeant le nombre de positions prédites le nombre de positions en input, et le décalage temporel. Bien que des réseaux de neurones classiques (dense) peuvent être utilisés de façon récursive, des architectures ont été désignées spécifiquement pour la prédiction de séries temporelles. En particulier, certaines ont déjà été appliquées à la prédiction de trajectoire de balle de tennis de table, comme les réseaux de neurones récurrents [119] (RNN), les réseaux de neurones « long-short term memory » [120] (LSTM), ou encore les modèles autorégressifs [121]. D'autres méthodes récurrentes comme les réseaux de neurones convolutionnels [122] (CNN), ou les « gated-recurrent

unit » [123] (GRU), sont également populaires pour la traduction des séries temporelles. La différence majeure entre ces architectures réside dans leur gestion des dépendances et de la mémoire [124]. En particulier, les GRU nécessitent moins de paramètres que les LSTM, ce qui facilite leur entraînement [124]. Cependant, l'approche récursive souffre d'erreurs cumulatives, ce qui réduit la précision au cours du temps [108], [120], [121], [125]. L'architecture générale des méthodes récursives est présentée Figure 2.10.

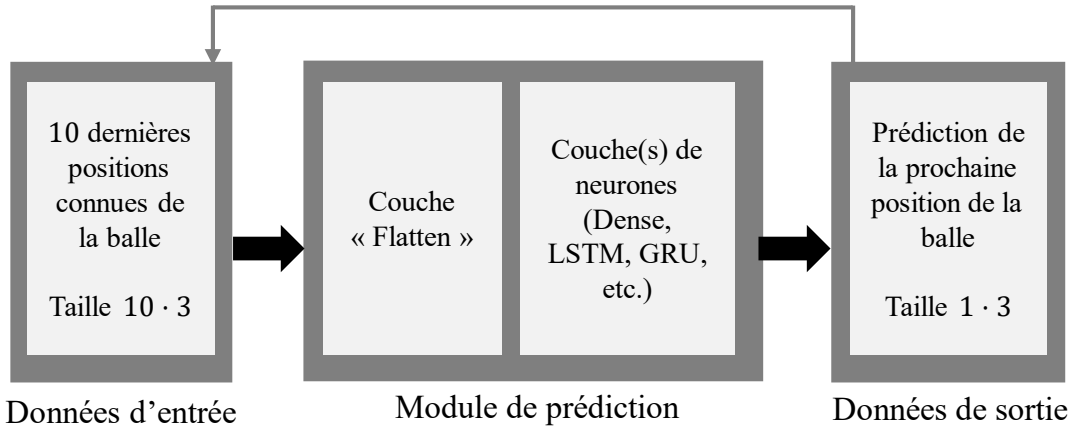


Figure 2.10 : Architecture générale des méthodes récursives

Enfin, les modèles « sequence to sequence » (seq2seq) évitent la dégradation de la précision au cours du temps en ne réutilisant pas les premières prédictions comme entrées pour les prédictions suivantes [126]. Ces modèles convertissent des séquences d'entrées en séquences de sorties, utiles par exemple pour la traduction, en utilisant une architecture « encodeur-décodeur », comme les autoencodeurs. Ces derniers compressent les données d'entrée avec un encodeur et reconstruisent ces données via un décodeur, ce qui les rend utiles pour la compression de fichiers ou encore l'extraction de features. Appliquées à la prédiction de trajectoires, ces features peuvent par exemple être la position initiale ou la vitesse. Lorsqu'ils sont utilisés pour la prédiction de trajectoire, les autosencodeurs sont adaptés en autoencodeurs de trajectoire, ou « trajectory autoencoders » [108] (TAE). Dans cette étude un masque a été appliqué pour gérer la taille variable des entrées, en remplaçant par des zéros pour les pas de temps où la position est inconnue. L'architecture générale des TAE est présentée Figure 2.11.

Dans [108], une méthode combinant apprentissage profond et un modèle génératif a été utilisé pour prédire les trajectoires d'une balle de tennis de table, en utilisant un autoencodeur variationnel « trajectory variational autoencoder » (TVAE). Contrairement aux TAE classiques, le TVAE

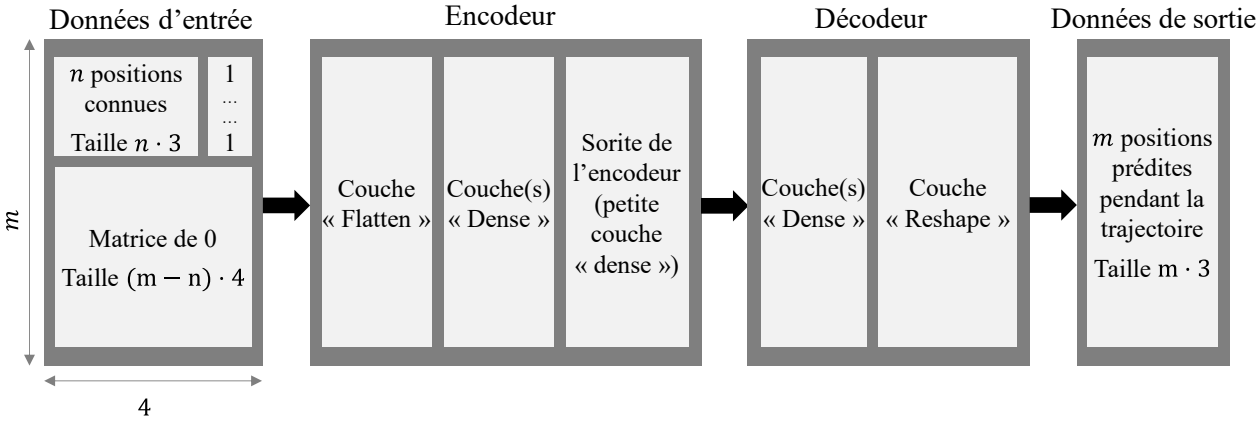


Figure 2.11 : Architecture générale des « trajectory autoencoders »

estime les incertitudes dans ses prédictions et fournit des intervalles de confiance. Cependant, il n'améliore pas la précision de la prédiction. Cette étude compare la précision d'un TVAE avec celle d'un modèle LSTM, montrant une meilleure précision sur les prédictions à long terme pour le TVAE, en dépit d'erreurs plus importantes à court terme.

Les modèles seq2seq traditionnels ne sont cependant pas optimisés pour la prédiction de séries temporelles. Ainsi des modèles seq2seq avancés utilisant des couches de neurones récurrentes – RNN, LSTM ou GRU – optimisées pour la prédiction de séries temporelles à la place des couches « dense » classiques, ont récemment été développés. Plusieurs exemples existent, comme les autoencodeurs LSTM [127], [128], ou les autoencodeurs GRU [129], [130]. Dans [131], un autoencodeur GRU (GRU-AE) a été développé pour la réduction de bruit et la détection d'erreurs dans les données de capteurs sur des véhicules. Un autre GRU-AE a été utilisé pour la prédiction du trafic [132].

Dans [133] et [126], un modèle seq2seq récurrent basé sur une architecture encodeur-décodeur (ED) a été proposé pour le traitement du langage. Ce modèle est constitué de deux RNNs, où le premier est utilisé pour encoder les entrées sur une longueur fixée, tandis que le second est utilisé pour décoder et retourner la séquence cible. Un réseau de neurones bidirectionnel basé sur un modèle seq2seq récurrent a été utilisé pour la prédiction de trajectoire de balle de tennis de table [134]. Cependant, les prédictions sont réalisées sur des séries de longueurs fixes – 5 positions sont prédites en utilisant les 13 observations précédentes. Dans [135], un autre modèle seq2seq récurrent, un GRU-ED a montré une meilleure précision à long terme, comparé au TVAE de [108], en utilisant des données simulées. Cette dernière étude a également comparé la précision obtenue

en utilisant des *transformers* – autre architecture seq2seq très populaire, basée sur un module d'attention [136] – pour la prédiction de trajectoires, basée sur [137]. Bien que performant pour des séquences complexes, ils ne sont pas à la hauteur pour des séquences homogènes de position de balles. Cependant, une limitation majeure de [135] est que le GRU-ED n'a été entraîné et testé qu'avec des données simulées.

Résumé :
Méthodes modèle dynamique

- Estimation du spin = défi majeur, impossible avec des caméras standards en temps-réel.
- Approches principales : observations avec caméras ultrarapides, analyse du mouvement du joueur, à partir de l'impact du spin sur la trajectoire

Méthodes d'apprentissage machine

- Identification de paramètres (+) Problème réduit (-) boîte noire
- Modèles récursifs (+) optimisés pour séries temporelles (-) Dégradation à long terme
- Modèles seq2seq (+) peu de dégradation (-) Modèles plus complexes, architectures pas optimisées pour séries temporelles

Gap : Modèles seq2seq pas encore optimisé pour prédiction de séries temporelles. Combinaison des méthodes d'apprentissage machine avec modèle dynamique peu explorée.

2.5 La stratégie de frappe

Tout système désirant jouer au tennis de table doit être en mesure de déterminer la configuration optimale permettant de renvoyer la balle à l'utilisateur de la manière la plus efficace possible. Ainsi, les robots capables de jouer au tennis de table intègrent un planificateur plus ou moins élaboré chargé de choisir le point de frappe, la vitesse et l'orientation de la raquette, et la configuration correspondante du robot au moment de la frappe.

2.5.1 Le choix du point de frappe

La méthode la plus populaire pour choisir le point de frappe est l'utilisation d'un plan de frappe virtuel (VHP). Cette méthode consiste à sélectionner le point de frappe comme l'intersection de la trajectoire prédite de la balle avec un plan virtuel prédéterminé. Cette approche permet de déterminer à la fois l'instant et la position de la frappe. Cette stratégie est utilisée notamment par les systèmes ayant des degrés de liberté redondants et un grand espace de travail [11], [12], [30]. En effet, en imposant un plan de frappe fixe, cette méthode réduit les possibilités : il peut arriver qu'aucune trajectoire réalisable ne soit trouvée pour certaines balles, alors qu'elles auraient pu être renvoyées en choisissant une autre position de frappe.

Le plan de frappe virtuel peut être vertical ou horizontal, et son orientation est souvent adaptée à l'architecture du système. Pour la majorité des systèmes situés derrière la table, le plan de frappe est généralement vertical et placé près de l'extrémité de la table [11], [30], [102], [111]. Le robot Forpheus, robot delta quant à lui situé au-dessus de la table, le plan de frappe est un plan horizontal [12], ce qui permet de frapper la balle juste après le rebond pour limiter son impact sur la trajectoire.

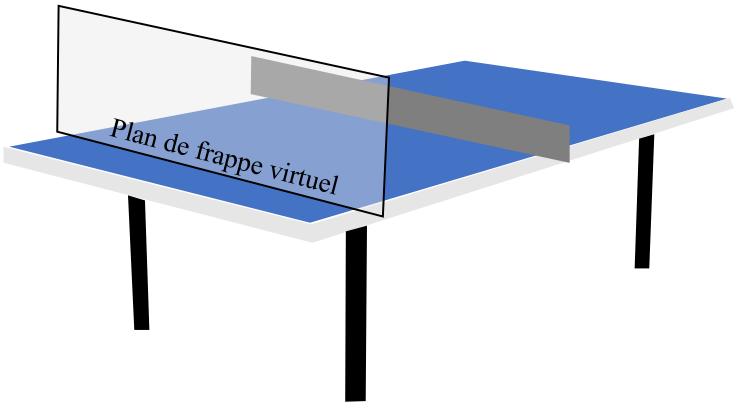


Figure 2.12 : Utilisation d'un plan de frappe virtuel

Lorsque l'utilisation d'un plan de frappe virtuel est trop limitante, les méthodes d'apprentissage machine peuvent être utilisés, en particulier les algorithmes d'apprentissage par renforcement. Dans [16], cette approche est utilisée pour choisir et adapter la commande parmi un ensemble de primitives de mouvement apprises lors d'une phase de préapprentissage. Dans [138], l'apprentissage par renforcement associe directement les mouvements du robot à la trajectoire prédite de la balle.

Enfin, lorsque les capacités physiques du système sont limitées, il peut être nécessaire d'itérer sur un ensemble de points de frappe possibles pour identifier le point de frappe qui minimise les efforts à effectuer par le robot. Dans [139], le point de frappe est par exemple choisi pour minimiser les déplacements du robot par rapport à sa position de base. L'inconvénient majeur de cette méthode réside dans la nécessité d'effectuer des calculs de cinématique inverse pour chaque point de frappe, ce qui augmente le temps de calcul.

2.5.2 La vitesse et l'orientation de la raquette

Une fois le point de frappe choisi, la vitesse et l'orientation de la raquette au moment de la frappe doivent être calculées pour permettre au robot de renvoyer la balle de l'autre côté du filet. Dans la majorité des systèmes, la balle doit être renvoyée sur un point précis de la table avec un temps de

vol donné. Pour cela, deux familles de méthodes sont généralement utilisées : les équations basées sur un modèle dynamique où l'apprentissage par renforcement.

Lorsque le modèle dynamique est utilisé, la connaissance du point visé lors du retour permet de déterminer la vitesse de la balle nécessaire juste après la frappe, grâce à l'équation (2).

Une fois les calculs effectués, le modèle de rebond entre la raquette et la balle est utilisé pour déterminer la vitesse et l'orientation de la raquette souhaitées au moment de l'impact [11], [12], [31]. Le modèle de rebond balle/raquette, qui a été largement étudié est similaire à celui du rebond table/balle [12], [38]. Il est détaillé dans les équations ci-dessous.

$$\begin{bmatrix} R^T & 0 \\ 0 & R^T \end{bmatrix} \begin{bmatrix} \dot{x}_1 - V \\ \omega_1 \end{bmatrix} = \begin{bmatrix} A_{vv} & A_{v\omega} \\ A_{\omega v} & A_{\omega\omega} \end{bmatrix} \begin{bmatrix} R^T & 0 \\ 0 & R^T \end{bmatrix} \begin{bmatrix} \dot{x}_0 - V \\ \omega_0 \end{bmatrix} \tag{11}$$

Avec, $\dot{x}_0$, ω_0, $\dot{x}_1$, ω_1 respectivement la vitesse et le spin de la balle avant et après la frappe. V est la vitesse de la raquette au moment de la frappe, R correspond à la matrice de rotation entre le repère inertiel et le repère relatif de la raquette. Elle est donnée par :

$$\boldsymbol{R} = \begin{bmatrix} \cos\alpha\cos\beta & -\sin\alpha & \cos\alpha\sin\beta \\ \sin\alpha\cos\beta & \cos\alpha & \sin\alpha\sin\beta \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

Avec α et β les angles représentant les rotations successives autour des axes z' et y' définissant l'orientation de la raquette. Enfin, A_{vv}, $A_{v\omega}$, $A_{\omega v}$, $A_{\omega\omega}$ représentent les matrices de transformation entre la balle avant et après la frappe. Elles sont définies comme suit :

$$\boldsymbol{A_{vv}} = \begin{bmatrix} -k_e & 0 & 0 \\ 0 & 1-k_v & 0 \\ 0 & 0 & 1-k_v \end{bmatrix}, \boldsymbol{A_{v\omega}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & k_v r \\ 0 & -k_v r & 0 \end{bmatrix}, \boldsymbol{A_{\omega v}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -k_\omega r \\ 0 & k_\omega r & 0 \end{bmatrix}$$

$$\boldsymbol{A_{\omega\omega}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1-k_\omega r^2 & 0 \\ 0 & 0 & 1-k_\omega r^2 \end{bmatrix}$$

Avec k_e le coefficient de restitution entre la balle et la raquette, k_v et k_ω les coefficients de frottement liés respectivement à la vitesse relative et au spin entre la balle et la raquette. En utilisant ces équations, la vitesse de la raquette V, et les angles représentant son orientation α et β peuvent être déterminés. [12]. La Figure 2.13 illustre ces paramètres.

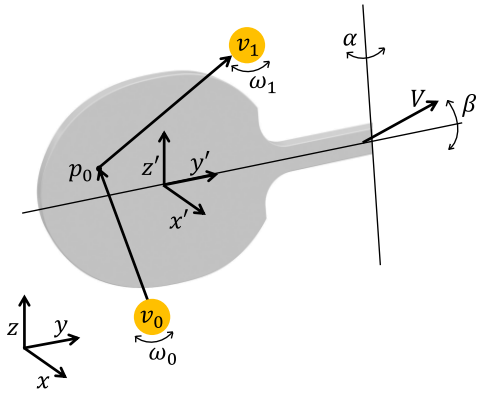


Figure 2.13 : Paramètres de la frappe

Dans le cas où le spin de la balle n'est pas considéré, un système de trois équations est obtenu pour cinq variables, rendant impossible l'obtention d'une solution unique. Dans ce cas, il est possible d'imposer deux contraintes supplémentaires permettant de résoudre le système d'équations. Dans [12], la vitesse verticale de la raquette v_z et l'angle d'élévation de la raquette β sont imposés, tels que $v_z = 0$ et $\beta = 0$. Ces paramètres ont été choisis pour leur impact faible sur la précision de la frappe.

L'alternative principale à l'utilisation du modèle dynamique repose sur les techniques d'apprentissage machine. Deux types d'approches principales existent, les approches utilisant des démonstrations [16], [25], [30], [35], [138], [140] et celles qui ne les utilisant pas [17], [27], [28].

L'apprentissage par imitation (IL) [141] est une méthode simple et stable permettant à un système robotique d'apprendre un comportement à partir de démonstrations. Dans [16], les auteurs présentent un système capable d'apprendre à jouer au tennis de table à partir de mouvements primitifs. Dans [30] et [13], deux systèmes capables d'apprendre à partir de démonstrations sous-optimales sont présentés. La principale limitation est que, l'IL requière un accès à de nombreuses démonstrations, ce qui est une procédure coûteuse.

L'apprentissage par renforcement (RL), quant à lui, permet d'apprendre de manière autonome par essai-erreur. Cependant, la phase d'exploration aléatoire initiale est difficile à appliquer sur des

systèmes physiques en raison des risques de dommages matériels. Plusieurs méthodes ont été développées pour le tennis de table [27], [140], mais le contrôle à haute fréquence reste l'un des plus gros challenges pour le RL [142]. Bien que des résultats prometteurs aient été obtenus en simulation, leur application à des systèmes physiques demeure complexe [143], [144], dû à la nécessité d'un contrôle précis et dynamique.

Dans [30], une approche combinant deux méthodes d'IL existantes [145], [146] a été implémentée. Cette approche est constituée de trois étapes principales : premièrement, un dataset de démonstrations variées est constitué. Une démonstration pouvant être utilisée consiste en une frappe de balle parvient à renvoyer la balle sur la partie adverse de la table. Il n'est pas nécessaire que la balle touche la table exactement à l'endroit visé. Deuxièmement, les données sont relabélisées comme si le point de chute de la balle lors du retour était celui effectivement visé [146]. Enfin, l'apprentissage se poursuit lorsque le système est en utilisation. Chaque fois qu'une balle est renvoyée à l'utilisateur, la trajectoire utilisée pour la frappe est labellisée et ajoutée au dataset. Cette procédure facilite l'amélioration constante du système au cours du temps. Malgré ces optimisations, le besoin initial en démonstrations reste élevé : 2480 démonstrations ont été nécessaires lors de la première phase d'apprentissage.

2.5.3 Cinématique inverse et choix de la configuration du robot

Après avoir calculé la position et la vitesse de la raquette, il est possible de réaliser la cinématique inverse du système pour obtenir les coordonnées articulaires correspondantes. Le processus de cinématique inverse utilisé dans le cadre des systèmes joueurs de tennis de table est rarement détaillé dans la littérature, car il est souvent intégré de manière native aux robots industriels.

Trois approches principales permettent de réaliser la cinématique inverse [147] : résolution analytique, les méthodes numériques basées sur la cinématique directe et les méthodes d'apprentissage.

Premièrement, la cinématique inverse peut être réalisée en résolvant analytiquement le système d'équations. Le problème à résoudre s'écrit sous la forme :

$$(q, \dot{q}) = f(x, \dot{x}) \tag{12}$$

Avec q et $\dot{q}$ les positions et les vitesses articulaires, tandis que x et $\dot{x}$ sont les vecteurs contenant les contraintes de position et de vitesse de la raquette. L'avantage principal de cette méthode est

qu'une fois la solution analytique trouvée, il est facile et rapide d'obtenir l'ensemble des solutions contenant les coordonnées et les vitesses articulaires. En revanche, plus le système est complexe, c. à. d. plus le nombre de degrés de liberté interne au système est important, plus il est difficile d'obtenir une solution analytique. Pour résoudre le système, il est nécessaire d'avoir un nombre de DDL égal au nombre de contraintes. Dans le cas de systèmes redondants, il est nécessaire d'ajouter d'autres contraintes à optimiser pour parvenir à une solution unique. De plus, cette méthode ne peut être généralisée à plusieurs architectures [147]. C'est pourquoi, en pratique, la résolution analytique de la cinématique inverse est rarement utilisée.

Deuxièmement, lorsque la cinématique inverse ne peut pas être résolue de façon analytique, une solution numérique peut être utilisée [148]. Cette méthode consiste à combiner les équations de la cinématique directe avec un algorithme d'optimisation itératif. Le principal avantage est que la cinématique directe est généralement plus facile à calculer, notamment en utilisant des matrices de transformation. En revanche, lorsque l'algorithme d'optimisation converge, il ne fournit qu'une solution possible. Pour obtenir l'ensemble des solutions, il est donc nécessaire de relancer plusieurs fois l'algorithme en variant les conditions initiales. De plus, les méthodes numériques étant itératives, elles sont généralement plus coûteuses en temps de calcul, ce qui peut être limitant pour des applications en temps-réel, comme le tennis de table [148]. Dans ce cas, il est nécessaire d'optimiser les hyperparamètres (pas d'apprentissage, nombre maximum d'itérations) et/ou le processus d'optimisation (Adam, Gradientdescent, etc.). Des outils logiciels comme Robotran [149] permettent également de résoudre la cinématique inverse numériquement, souvent sans nécessiter l'implémentation explicite de la cinématique directe ou de l'algorithme d'optimisation. Cependant, ces outils limitent souvent l'accès aux hyperparamètres, ce qui peut allonger le temps de calcul.

Enfin, les méthodes d'apprentissage sont une alternative pour apprendre la cinématique inverse. Cette approche est particulièrement utile lorsque le modèle cinématique n'est pas connu avec précision [150], [151]. De plus, la résolution de la cinématique inverse à l'aide de réseaux de neurones est rapide et permet une utilisation en temps réel, pour du contrôle adaptatif par exemple [152], [153], [154]. Il est utile de noter que les méthodes par apprentissage font également partie des méthodes itératives, mais que les itérations s'effectuent pendant la phase l'entraînement [155]. La principale limitation des méthodes d'apprentissage réside dans la difficulté des réseaux de neurones à prendre en compte l'ensemble des contraintes. Dans plusieurs travaux [156], [157],

[158], seule la position de l'effecteur est considérée, en négligeant son orientation. D'autres travaux négligent les limites physiques des moteurs (vitesses, accélération). En conséquence, la majorité des résultats obtenus jusqu'à présent concernent des simulations plutôt que des systèmes réels [151].

Résumé :

- Plan de frappe virtuel pour le choix du point de frappe très largement utilisé. (+) Aucun calcul supplémentaire (-) Limite les possibilités
- Alternative : itérer sur un ensemble de points de frappe possibles (+) explore l'ensemble des possibilités (-) Temps de calcul
- Choix d'un point cible pour le retour + Modèle de la frappe pour obtenir la pose et la vitesse de la raquette
- Cinématique inverse pour obtenir les positions et vitesses articulaires.
    - Résolution analytique au préalable (+) Rapide, ensemble des solutions (-) Pas toujours de solution analytique
    - Processus itératif (+) Plus facile (-) Donne solution unique, lent
    - Méthodes d'apprentissage par renforcement ou par imitation. (+) Processus unique, peu de calculs, pas besoin d'un modèle cinématique précis (-) Besoin de données de démonstration, effet boîte noire, solutions pas toujours exactes

2.6 Génération de trajectoire

Une fois l'ensemble des paramètres de la frappe déterminés, il est nécessaire de générer la trajectoire du robot entre sa position actuelle et sa position finale (au moment de la frappe). Étant donné que le tennis de table exige des mouvements à haute dynamique, les systèmes physiques sont souvent utilisés au maximum de leurs capacités. Ainsi, l'objectif principal de la génération de trajectoire est de trouver les mouvements les plus simples pour chacun des joints.

La méthode la plus couramment utilisée par les systèmes robotiques joueurs de tennis de table repose sur les polynômes quintiques. L'avantage des polynômes quintiques avec que la trajectoire de chaque moteur est générée de façon très rapide tout en garantissant un contrôle précis du mouvement à chaque pas de temps et en minimisant l'accélération et le jerk tout au long du mouvement [117]. Un polynôme quintique étant un polynôme de degré 5, il est possible d'imposer jusqu'à six contraintes sur chaque trajectoire : la position, la vitesse et l'accélération initiales du robot $q_0 = q(t_0)$, $\dot{q}(t_0) = v_0$ et $\ddot{q}(t_0) = a_0$, et la position, la vitesse et l'accélération du robot au moment de la frappe $q_d = q(t_d)$, $\dot{q}(t_d) = v_d$ et $\ddot{q}(t_d) = a_d$.

La trajectoire est représentée par le polynôme suivant :

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \quad (13)$$

On obtient alors un système linéaire à six contraintes et six inconnues :

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & t_d & t_d^2 & t_d^3 & t_d^4 & t_d^5 \\ 0 & 1 & 2t_d & 3t_d^2 & 4t_d^3 & 5t_d^4 \\ 0 & 0 & 2 & 6t_d & 12t_d^2 & 20t_d^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} q_0 \\ \dot{q}_0 \\ \ddot{q}_0 \\ q_d \\ \dot{q}_d \\ \ddot{q}_d \end{bmatrix} \tag{14}$$

permettant de calculer les coefficients a_i. Ce système est résolu indépendamment pour chaque moteur. En général, les accélérations initiale et finale $\ddot{q}_0$ et $\ddot{q}_\mathrm{d}$ sont fixées à 0 tout comme la vitesse initiale du robot $\dot{q}_0$. La même méthode peut également être utilisée pour générer la seconde moitié de la trajectoire, permettant au robot de revenir à sa position d'attente.

Une trajectoire peut également être générée en utilisant un algorithme d'optimisation, combiné au modèle dynamique du système, comme le contrôleur « iterative Linear Quadratic Regulator », (iLQR) [159]. Ce dernier génère une trajectoire en minimisant une fonction de coût arbitrairement définie. L'iLQR peut être appliqué aux systèmes non linéaires, étant donné son fonctionnement par itérations successives, en supposant une linéarité locale. Dans [49], un iLQR a été implémenté pour le contrôle d'un quadrotor. La fonction de coût à minimiser peut par exemple inclure des termes pour pénaliser le couple nécessaire pour les moteurs, les déplacements, et l'erreur à la position finale souhaitée, telle que :

$$J = \int_{t_0}^{t_f} \left((\boldsymbol{q} - \boldsymbol{q_0}) \boldsymbol{Q} (\boldsymbol{q} - \boldsymbol{q_0})^T + \boldsymbol{u} \boldsymbol{R} \boldsymbol{u}^T \right) dt + \left(\boldsymbol{q} - \boldsymbol{q_f} \right) \boldsymbol{Q_f} \left(\boldsymbol{q} - \boldsymbol{q_f} \right)^t \tag{15}$$

Avec $\boldsymbol{q}$ les coordonnées articulaires, $\boldsymbol{u}$ le vecteur de commande en couple, et $\boldsymbol{Q}$, $\boldsymbol{R}$ et $\boldsymbol{Q_f}$ trois matrices de poids.

Cependant, cette méthode présente plusieurs inconvénients : elle nécessite une connaissance précise du modèle dynamique du système et une grande capacité de calcul, rendant son application en temps-réel très difficile. De plus, pour les bras robotiques, la trajectoire obtenue avec un iLQR est souvent similaire à celle obtenue avec les polynômes quintiques.

Deuxièmement, pour les systèmes dotés de degrés de liberté redondants, il est utile d'imposer une trajectoire rectiligne à la raquette (Figure 2.14). En effet, cela permet de minimiser l'impact d'une erreur de positionnement de la raquette, ou d'une erreur dans la prédiction de la trajectoire de la balle. Dans ce cas, le robot adopte une position intermédiaire derrière le point de frappe dès que celui-ci est identifié, et les trajectoires des joints sont générées de manière à produire un mouvement rectiligne avec l'effecteur [12].

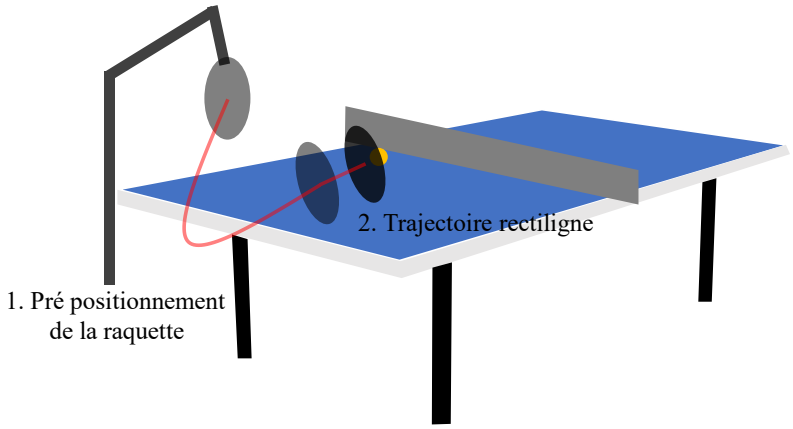


Figure 2.14 : Stratégie de frappe avec trajectoire rectiligne

Enfin, tout comme le choix du point de frappe et des contraintes d'orientation et de vitesse de la raquette, certains systèmes génèrent la trajectoire à partir de primitives et de démonstrations avec un algorithme de IL ou de RL [25].

Résumé :

- Utilisation des polynômes quintiques pour générer les trajectoires les moins contraignantes pour les moteurs.
- Contrôleurs incluant le modèle dynamique pas assez rapide pour du temps-réel (iLQR)
- Utilisation d'une position intermédiaire proche du point de frappe permet de limiter la quantité de mouvement au moment de la frappe.
- Lorsque c'est possible, trajectoire linéaire pour la raquette pour limiter l'impact des erreurs sur le retour.
- Lorsque l'IL ou le RL sont utilisés, la trajectoire peut être générée directement avec le mélange de primitives.

2.7 Contrôle

La dernière étape de l'algorithme consiste à implémenter la structure de contrôle permettant un suivi de trajectoire précis du système. La majorité des robots industriels et des cobots possèdent leur propre structure de contrôle interne. Dans ce cas, il n'est généralement pas nécessaire de réimplémenter le module de contrôle. Cependant, cela n'est souvent pas vrai pour les robots personnels.

La capacité des bras robotiques à suivre une trajectoire avec précision est primordiale dans de nombreux domaines d'applications. Ainsi, le développement de structures de contrôle pour le suivi de trajectoires est un sujet de recherche actif depuis plusieurs dizaines d'années [160].

Le suivi de trajectoire requiert soit un modèle dynamique précis du système, soit un algorithme de suivi agressif avec des gains élevés pour le feedback. Cependant, pour la majorité des bras robotiques, obtenir un modèle dynamique précis reste difficile en raison des non-linéarités (comme les frottements) et des incertitudes (par exemple, les paramètres inertiels) dans ces systèmes dynamiques [161]. Cela complique l'implémentation de ces approches. L'incapacité des servocontrôleurs de gérer ces non-linéarités et ces incertitudes mène à une dégradation dans la précision du suivi de trajectoires [162].

Dans l'industrie, la solution consiste généralement à faire un compromis entre la précision du suivi de trajectoire et la réduction de la fréquence d'opération [163]. Ce problème est accentué par le développement des bras robotiques bon marché, souvent fabriqués par impression 3D, qui présentent des frottements plus importants et des moteurs moins puissants.

L'architecture de contrôle la plus répandue consiste en un contrôle en couple avec un contrôleur linéaire, tel que le traditionnel contrôleur proportionnel-intégral-dérivé (PID). Pour compenser le temps de réponse du contrôleur, une compensation feedforward, calculée à partir du modèle dynamique est habituellement ajoutée. D'autres alternatives consistent à utiliser des modèles simplifiés [164], ou des méthodes d'apprentissage machine [161]. En particulier, les méthodes itératives, comme l'*iterative learning control* (ILC), exploitent la répétition des tâches et permettent un suivi de trajectoire efficace. Bien qu'elles ne nécessitent pas de modèle dynamique précis [165], [166], l'incorporation de celui-ci peut améliorer les performances [163]. Cependant, si ces méthodes sont particulièrement efficaces pour les tâches répétitives, comme le travail sur les lignes d'assemblage, leur généralisation à de nouvelles trajectoires reste limitée [167].

Les structures de contrôle basées sur les réseaux de neurones (NNs) ont suscité un intérêt croissant grâce à leur capacité à généraliser au-delà des données d'entraînement [163]. En particulier, ce potentiel d'approximation des réseaux de neurones a été utilisé dans des structures de contrôle pour le suivi de trajectoire, en apprenant la dynamique directe ou inverse des systèmes [168], [169], [170], [171]. Selon [163], lorsqu'une architecture basée sur des réseaux de neurones est utilisée, le problème de contrôle et soit un problème d'apprentissage par renforcement [172], [173], de contrôle adaptatif [174], où de contrôle optimal [175].

Différentes architectures de réseaux de neurones, telles que les réseaux récurrents (RNN), feedforward ou à fonctions de base radiale (RBF), ont été utilisées pour approximer la dynamique directe [171], [176], [177], [178]. Dans [179], les réseaux de neurones ont été utilisés pour apprendre la dynamique inverse, pour être ensuite intégrés comme compensation feedforward. Il a été démontré que les RNN et les LSTM surpassent d'autres techniques telles que les processus gaussiens, pour l'apprentissage de la dynamique inverse quand suffisamment de données sont disponibles pour l'entraînement [180], [181]. Cependant, la plupart de ces études utilisent le contrôle en couple. Par exemple, pour un bras à sept DDL, un NN avec environ 10^5 paramètres est souvent requis, ou bien plusieurs réseaux pour modéliser différentes parties du système dynamique [182]. Ces approches utilisant le contrôle en couple présentent également un écart important entre les performances en simulation et en réalité. Dans [183], un contrôleur adaptatif basé sur des réseaux de neurones a été utilisé pour le suivi de trajectoire d'un robot manipulateur, mais seuls des résultats simulés sont présentés.

Si le design d'un contrôleur joue un rôle central dans les structures de contrôle, il est important de noter qu'une structure de contrôle doit également répondre à d'autres questions : « Quelles variables doivent être contrôlées et mesurées, et quelles entrées doivent être manipulées ? » « Quelles configurations de contrôle doivent être utilisées (contrôle en cascade, feedforward, etc), comment doivent-elles être organisées, combien de degrés de liberté le contrôleur nécessite-t-il pour parvenir à la performance désirée ? » [184].

Dans [162] et [185], une autre architecture de contrôle a été proposée. Des structures basées sur des réseaux de neurones ont été utilisées dans une boucle externe (par exemple pour un contrôle en position) pour une compensation feedforward, sur un quadrotor avec un réseau de neurones profond, et pour un robot à 7 degrés de liberté avec un RNN. Ces études mettent en évidence trois

avantages principaux de l'intégration d'une compensation feedforward basée sur des réseaux de neurones avec un contrôleur proportionnel-dérivé (PD) en position.

Premièrement, les architectures de réseaux de neurones peuvent être appliquées à des systèmes variés présentant des dynamiques complexes tout en assurant leur stabilité [185]. Deuxièmement, il s'agit d'une approche facile à mettre en œuvre. Elle ne requiert aucune connaissance préalable sur le système et peut être appliquée directement à des trajectoires inconnues, sans processus d'adaptation [163], [170]. Troisièmement, avec un entraînement approprié, cette méthode offre une précision élevée dans le suivi de trajectoire, tout en limitant les besoins de calcul grâce à l'utilisation de réseaux de neurones de taille réduite [162].

Dans [162], l'apprentissage de la dynamique d'un robot Baxter flexible a été réalisé avec un contrôle en position en combinant un contrôleur PID avec un RNN bidirectionnel ou un RNN associé à un algorithme d'ILC, tous deux avec environ 10^3 paramètres, intégrés dans un seul modèle. Cette approche a permis de réduire les erreurs de suivi de trajectoire de 12 % à 54 % par rapport au contrôleur PID de référence, sur des trajectoires aléatoires.

Dans [163], l'apprentissage de la dynamique inverse d'un bras robotique industriel à sept DDL a été mené en simulation. Un algorithme de contrôle a été implémenté pour collecter les données nécessaires à l'entraînement du réseau de neurones sur des trajectoires variées. Cependant, cette méthode exige une simulation de la dynamique de grande qualité, ce qui est rarement possible pour les robots non industriels, où le modèle dynamique n'est pas toujours connu avec précision. Dans cette étude, un réseau de neurones avec environ $2\ 10^4$ paramètres – organisé en deux couches de 100 neurones – a été utilisé comme compensation feedforward pour les erreurs dynamiques du contrôleur interne en position d'un robot ABB IRB6640-180, à 6 DDL. Cette approche a réduit les erreurs de suivi de 80 % à 90 % par rapport à l'absence de compensation, avec une erreur quadratique moyenne (RMSE) de 2,145° sur l'ensemble des tests. Contrairement à une étude précédente des mêmes auteurs, qui utilisait un réseau de neurones unique pour l'ensemble du système, les auteurs ont expliqué que sur le robot ABB utilisé, la dynamique des joints était découplée - l'impact du mouvement d'un joint sur la dynamique des autres joints est négligeable. Ce découplage permet d'entraîner six réseaux de neurones séparément, chacun approximant la dynamique inverse d'un joint. Cependant, seuls les résultats pour un réseau global, regroupant tous les joints, ont été présentés.

La Figure 2.15 résume les architectures de contrôle pour le suivi de trajectoire

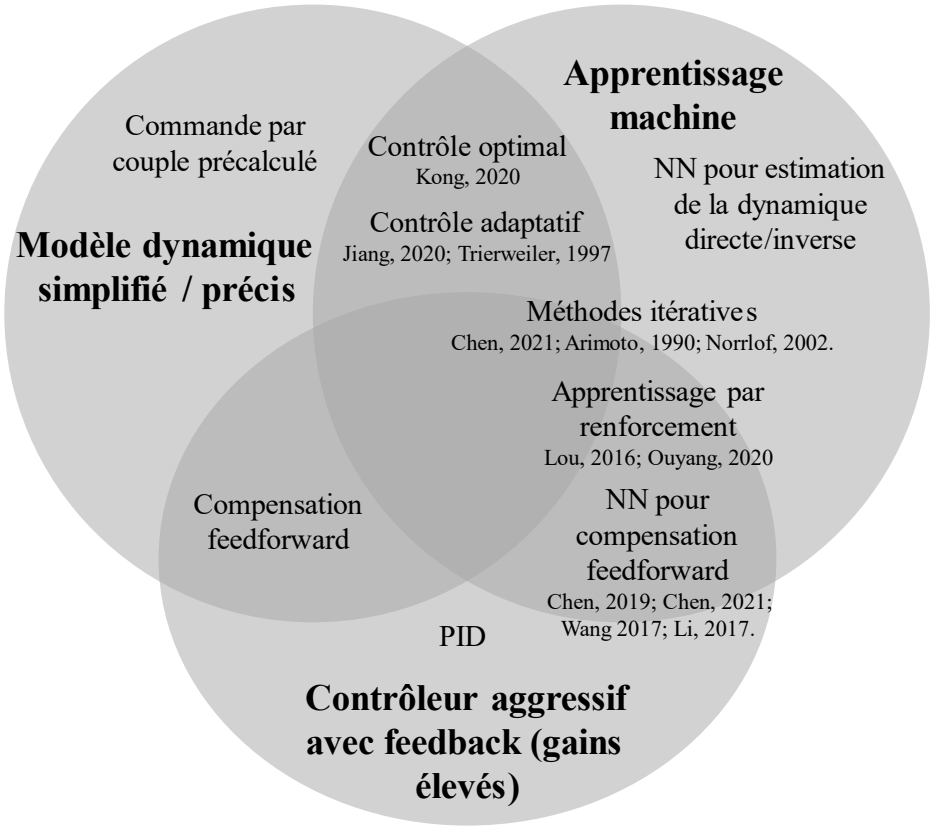


Figure 2.15 : Résumé des structures de contrôle existantes

Résumé :

- Capacité des robots à suivre une trajectoire dynamique, primordiale. Deux approches majeures : utilisation modèle dynamique ou contrôleur agressif en boucle fermée
- Modèle dynamique précis très difficile à obtenir en pratique pour des robots non industriels
- Contrôleurs agressifs : temps de réponse = erreur dans le suivi pour les trajectoires dynamiques
- Méthodes itératives = suivi de qualité, mais difficile de généraliser à des trajectoires inconnues
- Utilisation de NN possible pour l'apprentissage du modèle dynamique
    - Contrôle en couple = beaucoup de paramètres, sources d'erreurs importantes, beaucoup de données d'entraînement nécessaires
    - Contrôle en position : possibilité d'utiliser un NN comme compensation feedforward, moins de paramètres. Méthode peu explorée.

Gap : Pas de structure de contrôle simple à mettre en œuvre sans connaissance préalable sur le modèle dynamique. Utilisation de NN pour compensation feedforward semble prometteur, mais aucun mode d'emploi systématique pour une utilisation optimale

CHAPITRE 3 MOTIVATION SCIENTIFIQUE

3.1 Motivation

Le marché mondial de la robotique est en pleine expansion porté par l'intégration de systèmes robotiques avancés et de l'IA, qui transforment de nombreux secteurs. En particulier, le développement des robots collaboratifs, ou cobots, offre de nouvelles opportunités. Cependant, leur coût élevé constitue un frein majeur à leur adoption. Par ailleurs, les robots personnels souffrent de performances limitées. Ils sont moins rapides et moins précis, avec une charge maximale et des espaces de travail restreints, ce qui limite leurs applications.

L'utilisation de l'IA et de l'apprentissage machine augmente les capacités des cobots et robots personnels, ouvrant ainsi de nouvelles opportunités pour les systèmes robotiques. Associée aux nouvelles techniques d'impression 3D, l'IA devrait permettre de réduire l'écart de performance entre les systèmes robotiques personnels abordables et les cobots traditionnels, notamment en matière de dextérité, rapidité et précision.

La motivation de cette thèse est donc de démontrer les performances que peuvent atteindre les bras robotiques de bureau intégrant l'intelligence artificielle.

3.2 Problématique

3.2.1 Problème principal

Comme nous l'avons vu dans la section 2.1, le tennis de table est souvent utilisé comme banc d'essai pour tester et démontrer les performances des systèmes robotiques. Ce choix s'explique par les contraintes de détection en temps-réel, de prise de décision rapide, de contrôle précis des mouvements et de design. Bien que de nombreux systèmes aient déjà été développés, avec des architectures variées, la section 2.2 a montré que tous utilisent des cobots avec des composants industriels, voire des robots industriels complets, avec une redondance dans les degrés de liberté, entraînant des coûts élevés. Les robots de bureau sont quant à eux cantonnés à des applications exigeant un niveau de performance moins élevé. Les bras robotiques de bureau qui intègrent des méthodes d'intelligence artificielle sont émergeants. À notre connaissance, il manque de documentation systématique sur leurs gains de performance.

3.2.2 Sous-problèmes

Dans la section précédente, l'état de l'art a permis d'identifier les manquements actuels dans la littérature concernant le tennis de table avec des bras robotiques de bureau :

1. Les modèles seq2seq récurrent n'ont pas été appliqués à la prédiction de la trajectoire de la balle.

La capacité du système à prédire la trajectoire de la balle, et en particulier son spin, est cruciale pour atteindre un haut niveau de performance. Comme démontré dans la section 2.5, de nombreuses méthodes basées sur l'apprentissage machine ont été développées, mais restent sous-optimisées. En particulier, les modèles seq2seq récurrent, spécialisés pour la prédiction de séquences temporelles, n'ont pas encore été appliqués pour la prédiction de trajectoire dans le contexte du tennis de table.

2. Aucune méthode hybride, combinant modèle dynamique et apprentissage machine n'a été développée pour la prédiction de la trajectoire de la balle.

Concernant toujours la prédiction de la balle, la section 2.5 à mis en évidence que bien que de nombreuses approches reposant sur les équations des modèles dynamiques ou sur des algorithmes d'apprentissage machine aient été proposées, aucune méthode hybride ne combine les algorithmes les plus performants des deux approches. Une telle méthode pourrait tirer parti des forces de chaque approche pour améliorer la précision des prédictions.

3. Il n'existe pas de structure de contrôle systématique et rapidement déployable pour le suivi de trajectoire à haute dynamique, sans connaissance préalable du modèle dynamique.

Un suivi précis des trajectoires lors de la frappe est également un facteur déterminant pour les performances globales. La section 2.8 a mis en lumière que si les robots industriels et des cobots disposent souvent de structures de contrôle interne basées sur les équations des modèles dynamiques, ce n'est pas le cas des robots personnels. De plus, déterminer précisément les paramètres du modèle dynamique de ces robots est complexe. Il existe donc un intérêt réel à développer des structures de contrôle ne nécessitant aucune connaissance préalable du modèle dynamique. Cependant, il n'existe pas aujourd'hui de structure de contrôle de ce type, ayant une mise en œuvre rapide et systématique, permettant un suivi précis de trajectoires à hautes dynamiques.

4. Les algorithmes développés pour le tennis de table ne sont pas adaptés aux problématiques des bras robotiques de bureau.

L'utilisation de bras robotiques de bureau pour le tennis de table exige des ajustements spécifiques des algorithmes. En particulier, les sections 2.6 et 2.7 ont montré que les systèmes basés sur des cobots ou des bras industriels peuvent se permettre des simplifications arbitraires. Il peut s'aigr par exemple de la réduction du nombre de configurations analysées, un point de frappe minimisant l'impact du spin, ou améliorant la robustesse de la frappe. Ces simplifications ne sont pas toujours envisageables pour les bras robotiques de bureau, qui nécessitent des algorithmes spécifiquement adaptés afin de préserver des performances optimales. À ce jour, aucun algorithme ne prend en compte les spécificités propres à ces robots.

3.3 Objectif général

L'objectif général de cette thèse est donc de **développer un système collaboratif de tennis de table basé sur un bras robotique de bureau**, afin de démontrer les performances atteignables par des bras robotiques de bureau, intégrant l'intelligence artificielle.

3.4 Objectifs spécifiques

En particulier, cet objectif général sera rempli en répondant aux objectifs spécifiques suivants au cours du développement : développer une méthode de prédiction de trajectoire basée sur un modèle seq2seq récurrent (SO1), concevoir une méthode de prédiction de trajectoire hybride combinant l'IA avec des approches traditionnelles basées sur le modèle dynamique (SO2), développer une structure de contrôle minimale, sans connaissance préalable du modèle dynamique, utilisant des réseaux de neurones (SO3), et développer un algorithme pour le tennis de table collaboratif adapté spécifiquement aux contraintes des bras robotiques de bureau (SO4).

CHAPITRE 4 ARTICLE 1 : REAL-TIME TRAJECTORY PREDICTION OF A PING-PONG BALL USING A GRU-TAE

Published in *Applied Intelligence*, on January 17, 2025.

Abstract: Table tennis with collaborative robots has been a challenge in robotics for decades, due to its unique challenges, primarily requiring precise real-time ball trajectory predictions to enable responsive, accurate gameplay. Traditional physical models, while widely studied, struggle with factors like spin, limiting accuracy. With advancements in computational power, data-driven approaches as sequence-to-sequence (seq2seq) models as trajectory autoencoders (TAE) offer potential for improved long-term prediction. However, they are not fully optimized for time-series prediction. More advanced seq2seq models with recurrent layers are better suited and improve prediction accuracy but remain underexplored for trajectory prediction. Additionally, recurrent layers integrated with TAE, as gated-recurrent unit TAE (GRU-TAE), have not been applied for real-world trajectories.

This study introduces a novel GRU-TAE model designed for high-accuracy, low-latency trajectory predictions in table tennis. Our approach was evaluated on both real and simulated data, demonstrating that TAE-based architectures outperform traditional recurrent models (e.g., LSTM, GRU, RNN) by 44\% in long-term prediction accuracy, achieving an average computation time of 8.2 ms. Additionally, GRU-TAE improves accuracy by 14\% over traditional TAE models and by 41\% compared to baseline model-based methods in predicting real ball trajectories. These results are important because they enhance robotic arm performance in returning balls to human players, allowing earlier positioning for ball return, and reducing late position adjustments.

This work sets the foundation for integrating GRU-TAE with advanced model-based approaches, aiming toward real-world deployment in collaborative table tennis robots. **Keywords:** robotics, machine learning, neural networks, trajectory prediction, table tennis

4.1 Introduction

Table tennis with collaborative robots has been a long-time topic of robotics research due to its inherent challenges promoting the integration between robotics and AI for many problems, in real-time sensing, intelligent decision-making, motion control and robotic design. Since the development of the first ping-pong player robot in 1988 [1], various robotic systems have been

developed using industrial serial robotic arms [2], [3], [4], Cartesian systems [5], [6], parallel robots [7] and even humanoid systems [8], [9].

Accurately predicting the incoming ball's trajectory in real-time is an essential challenge for these robots, relying on prior visual observations [10]. These visual measurements are often noisy, containing outliers and sparse data [11]. The accuracy of ball trajectory prediction is directly linked to the robot's performance level. Consequently, table tennis robotic systems often includes a high detailed prediction method [3], [7]. The chosen prediction method also impacts other modules of the table tennis algorithm, particularly the targeted hitting zone. Moreover, the latency of these predictions plays a crucial role, as the computation must be significantly faster than the ball's speed [12]. Beyond table tennis, dynamic highspeed robotics commonly require real-time predictions of future physical quantities based on previous measurements.

Methods for predicting the trajectory of a ping-pong ball have been extensively studied and can be categorized into two main approaches detailed in the following paragraphs: model-based [7], [10], [13] and data-driven methods [4], [12], [14]. Figure 1 summarizes these methods.

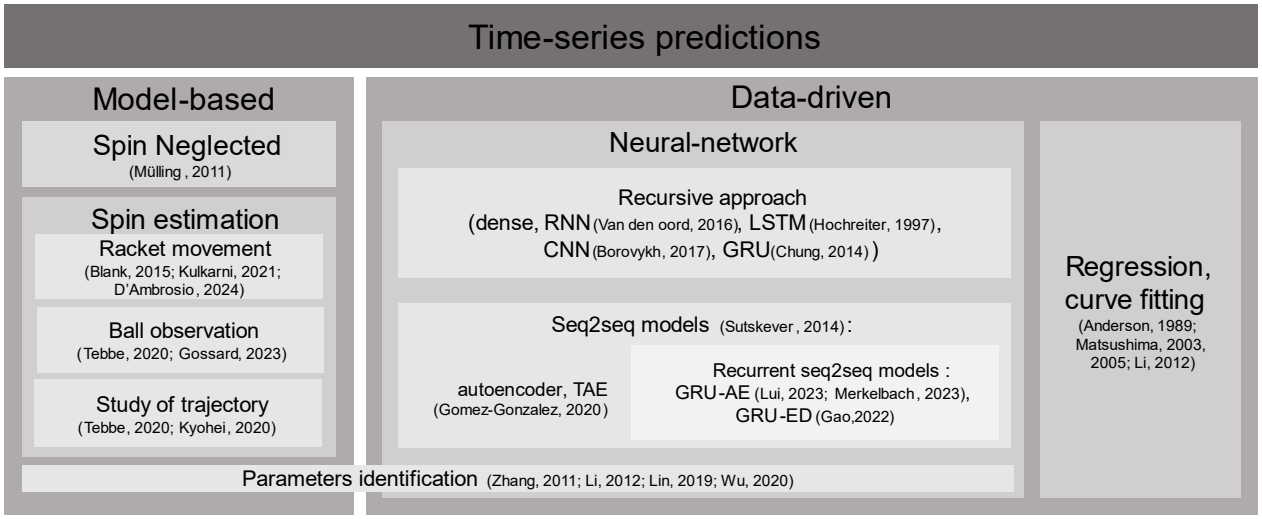


Figure 4.1 : Performance of trajectory tracking with various NNs for (A) 3-DOF robotic arm, fast trajectories.

4.1.1 Model-based methods

Model-based methods are often preferred for modeling and predicting trajectories in high-speed robotic systems due to their computational efficiency and grounding in well-established physical

models, known for their accuracy [13], [15]. Although these methods rely on differential equations, they can be computed relatively quickly.

Model-based methods typically involve two main steps. First, the position, velocity, and sometimes the spin of the ball are estimated based on observations. Then, the trajectory is predicted by solving the system of differential equations derived from the dynamic model [14].

However, a key limitation is that certain critical parameters remain difficult to estimate accurately, even with detailed knowledge of physical models [8]. In table tennis, for example, spin is a crucial factor, necessary for human players to win competitions. In fact, spin significantly affects the ball's trajectory, especially after it bounces on the table [10]. Accurately calculating ball spin in real-time from standard camera images, however, is not feasible [7].

Due to the challenges in measuring ball spin, some approaches opt to omit spin from trajectory predictions [13]. However, neglecting ball spin results in a significant decrease in performance. To compensate for this, ping-pong robots that ignore spin typically try to hit the ball immediately after its bounce [7] or instruct users to minimize spin during play.

Several methods have been explored to incorporate the ball's rotation speed into dynamic models, and these can be classified into three main approaches. The first, involves estimating spin based on the racket's movement during the stroke [16], [17]. Although some studies have successfully classified strokes using the racket's movement, they have not achieved precise spin estimation. The second approach, compute the spin directly by observing a logo or a drawn pattern on the ball [10], [11]. This method, however, requires expensive high-speed, high-resolution cameras and involves significant computational complexity. The third approach analyzes the ball's trajectory to infer spin [7], [10]. However, these last two approaches are highly sensitive to the accuracy of the dynamic model parameters. Given these additional challenges, model-based methods that include spin estimation are not considered in this study.

4.1.2 Data-driven methods

Data-driven methods can consider the ball spin by considering its impact on the ball's trajectory. The main advantage of data driven methods is that they do not require any information on the equations of the dynamic model. In [18], [19], [20], the dynamic model and the force analysis were not considered; instead, trajectory prediction was performed by mapping from previous

observations using regression methods, such as polynomial regression or local linear regression. In [21], a second-order polynomial curve fitting method was applied, where the observed trajectory could consist of ball positions either before or after bouncing on the table. However, since the dynamic model of the ball is highly non-linear, these methods tend to produce significant systematic errors [12].

4.1.2.1 Neural network-based methods

Research on data-driven and machine learning methods has notably increased, and these techniques have been widely applied to predict ping-pong ball trajectories, using various inputs and outputs. The three main approaches are parameter identification, recursive architectures, and sequence-to-sequence models, as described here below.

First, parameters identification focuses on learning specific parameters of the trajectory. In [21], two neural networks are employed to determine the polynomial coefficient of the trajectory before and after the ball's rebound. A similar method is presented in [22]. In [6], a neural network is trained to directly learn the hit point between the ball and the racket, using only 4 observations of the ball. To predict the bounce position, [23] developed a convolutional neural network (CNN). However, the accuracy of these methods is suboptimal because certain parameters remain unknown.

Secondly, recursive approaches compute the successive positions of the ball directly throughout its trajectory. The method uses a neural network to predict the next position based on previous observations. The new prediction is then fed back as input for the following prediction. This recursive process allows for the prediction of the entire trajectory, iteratively. The algorithm can be adapted by varying the number of predicted positions, input positions, and time shifts. While dense neural networks can recursively predict ball trajectories, specialized recurrent neural architectures designed for time series modeling have also been employed. For instance, recurrent neural networks (RNNs) [24], long short-term memory (LSTM) networks [25], and autoregressive models [26] have been used for ping-pong ball trajectory prediction. Other recurrent methods including convolutional neural networks (CNNs) [27] and gated-recurrent unit (GRU) networks [28] are also widely used for time series prediction. The key distinction between RNNs, LSTMs, and GRUs lies in how they handle memory and dependencies [29]. Notably, GRU networks have fewer parameters than LSTMs, making the training process easier and faster [29]. However,

recursive approaches suffer from cumulative errors, which reduce accuracy over time [12], [25], [26], [30].

Thirdly, "Sequence to sequence" (seq2seq) models avoid the issue of prediction degradation by not using previous predictions as future inputs [31]. These models convert input sequences into output sequences, such as in translation tasks, often using an encoder-decoder architecture like autoencoders. An autoencoder compresses input data via an encoder and reconstructs it through a decoder, making it useful for tasks such as file compression and feature extraction. For trajectory prediction, such features might include initial position, speed and spin.

For trajectory prediction, autoencoders have been adapted into trajectory autoencoders (TAE) [12]. In this study, a zero-padded mask is used to handle variable input sizes by replacing missing data with zeros for time steps where no information is available. In [12], a method combining deep learning with a conditional generative model is employed to predict ping-pong ball trajectories using a trajectory variational autoencoder (TVAE). Unlike a standard trajectory autoencoder (TAE), the TVAE estimates prediction uncertainties and provides confidence intervals. However, it does not improve the accuracy of the predictions. The study compares the accuracy of the TVAE with an LSTM model on 200 simulated and 35 real trajectories, showing better long-term predictions for the TVAE despite greater short-term errors, during the first 10 steps of prediction [12].

Traditional seq2seq models, however, are not optimized for time-series prediction. Therefore, recurrent seq2seq architectures, using recurrent layers such as RNN, LSTM or GRU – designed specifically for time-series prediction—have been developed, replacing dense layers in the encoder and decoder. Examples include LSTM-autoencoders [32], [33] and GRU-autoencoders (GRU-AE) [34], [35]. In [36], a GRU-AE was used for noise reduction and fault detection in recordings from car sensors. Similarly, in [37], another GRU-AE was used to predict traffic flow. In [38] and [31] a recurrent seq2seq architecture based on an encoder-decoder (ED) model was proposed for language processing. It involves using two different RNNs, where the first encodes the inputs into a fixed-length code, and the second decodes it into the target sequence. In [39], a bidirectional NN using a recurrent seq2seq architecture was employed to predict real trajectories of a tennis ball. However, these predictions were limited to fixed-length time series, where 5 future ball positions were predicted from 13 prior observations.

In [14], another recurrent seq2seq architecture, a GRU-ED, was developed for ping-pong ball trajectory prediction. The results showed better long-term prediction compared to the TVAE from [12], using 1200 ball trajectories generated in a simulated environment for training, with RMSE values of 27 mm and 52 mm for the GRU-ED and the TVAE, respectively. This promising result illustrates the interest of using recurrent seq2seq architecture in table tennis trajectory prediction. This study also compared the accuracy of transformers – a popular seq2seq architecture based on attention mechanisms [40] – for trajectory prediction, as described [41]. The results indicated that transformers are particularly effective for sequences with complex dependencies but are less effective for homogeneous ball sequences [14]. However, a limitation of this paper is that the GRU-ED was only trained and evaluated using simulated data. To the best of our knowledge, it is the only study to apply a recurrent seq2seq model for ping-pong ball trajectory prediction.

In summary, based on the state-of-the-art, **combining a TAE with recurrent layers in a single architecture could improve the accuracy of ping-pong ball trajectory prediction, thereby enhancing the performance of real robotic ping-pong systems.** However, there is limited information on the accuracy of recurrent seq2seq models for trajectory prediction, and no result with real-world data. Moreover, recurrent seq2seq models combined with TAE architectures, such as GRU-TAE, have yet to be implemented. Lastly, no comparison has been made between seq2seq models and model-based methods with accurate dynamic equations.

The objective of this study is to use a GRU-TAE model designed for high-accuracy, low-latency trajectory predictions in table tennis. A subsequent objective is to compare the accuracy of GRU-TAE with traditional time-series prediction methods, including model-based approaches, recurrent models, and TAE. To meet this objective, both real and simulated ping-pong ball trajectories, with and without noise, were studied.

4.2 Methodology

The method section is organized as follows: First, the implementation of the GRU-TAE is described (Section 2.1). Secondly, experimentation setup is presented and architectures compared with the GRU-TAE are listed (Section 2.2). Thirdly the process used for building the datasets is explained (Section 2.3). Finally, the comparison strategy is outlined (Section 2.4).

4.2.1 Implementation of the GRU-TAE

The architecture of our GRU-TAE is inspired by the TAE architecture from [12], and is presented in Figure 2. The main difference between the TAE from [12] and the GRU-TAE is that the dense layers of the encoder and decoder are replaced by GRU layers.

To evaluate the performance of the architecture, prediction accuracy was tested using different hyperparameters, including learning rate, batch size, the amount of training data and the size and number of layers. The final accuracy is reported as the average RMSE of the best-performing architectures. Detailed results for the optimal hyperparameters can be found in the Appendix.

Interestingly, while traditional GRU-autoencoder designs replace the dense layers with GRU layers in both the encoder and the decoder, we found that the GRU-TAE achieved better accuracy when GRU layers were used only in the encoder. In this approach, the GRU cells extract significant parameters from the trajectory, which are then used to reconstruct the trajectory (which is no longer a time series) using dense layers.

The optimal architecture of the GRU-TAE is illustrated in Figure 2. The encoder of the GRU-TAE consists of two GRU layers: the first layer contains between 16 and 64 cells, and output a sequence, while the second layer is another GRU layer with 16 cells. The final output of the encoder is a vector of 16 values, with a total of between 2688 and 17376 trainable parameters. The decoder comprises two dense layers, one with up to 400 units and the other with 270 units, for a maximum

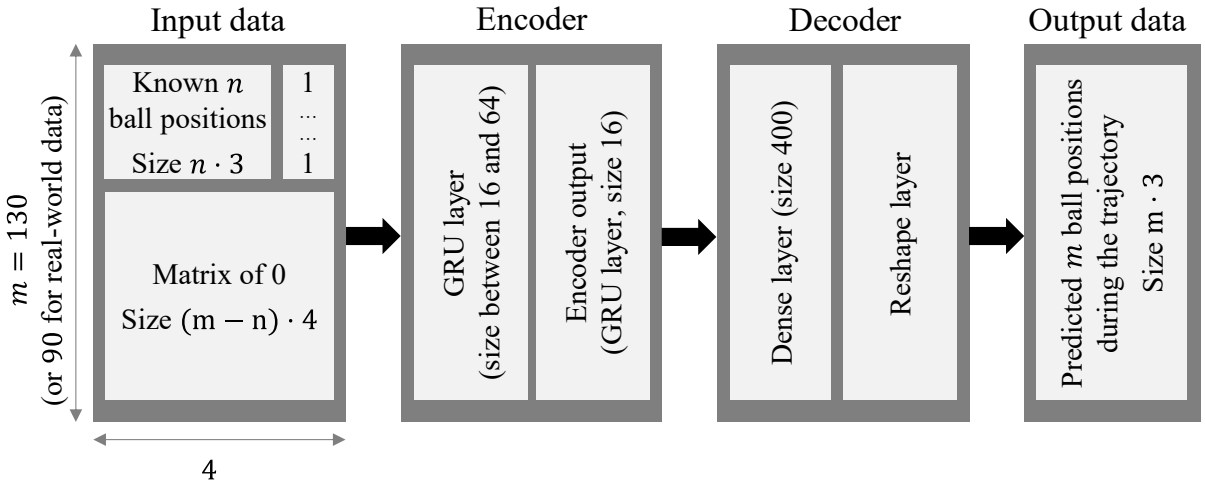


Figure 4.2 : Architecture of the GRU-TAE for ping-pong ball trajectory prediction.

of 115070 trainable parameters. Finally, the number of parameters across tested configurations does not exceed 132,446.

4.2.2 Experimentation setup

All data-driven architectures were implemented using Tensorflow [42] and Keras libraries [43] in Python, and training was conducted on GoogleColab servers. To optimize computation time for the tested methods, the code was precompiled using the Numba library [44], allowing for faster execution.

An open-source implementation of the code, including the training procedures for all tested architectures, is available at [45]. The compared architectures are listed in Table 1, with detailed implementation and training procedures provided in the Appendix.

Tableau 4.1 : List of time-series prediction methods used for comparison.

Approach	Architecture	Detail of layers
Model-based, no spin.		
Recurrent	Dense	Flatten, Dense(s)
Recurrent	LSTM	LSTM(s)
Recurrent	GRU	GRU
Recurrent	Convolutional	Conv1D
Recurrent	RNN	RNN
Autoencoder	TAE	Flatten, Dense(s), Reshape
Autoencoder	LSTM-TAE	LSTM, Dense(s), Reshape
Autoencoder	GRU-TAE	GRU, Dense(s), Reshape

4.2.3 Trajectory dataset construction

To evaluate the accuracy of the compared methods, both simulated (with and without noise) and real ball trajectories were analyzed.

First, for the simulated ball trajectories, the equations of the dynamic model of the ball were used to generate trajectories. These equations have been widely studied and are used in various works [7]. They are presented in equations (3) (4) and (5). Equation (3) describes the flying phase of the ball while equations (4) and (5) handle the rebound phenomenon, as outlined in [46].

Equation (3) describes the flight phase:

$$m\ddot{\boldsymbol{x}}(\boldsymbol{t}) = -m\boldsymbol{g} - m\ddot{\boldsymbol{x}}_{\boldsymbol{D}}(t) + m\ddot{\boldsymbol{x}}_{\boldsymbol{M}}(t) = -m\boldsymbol{g} - C_D\|\dot{\boldsymbol{x}}(t)\|\dot{\boldsymbol{x}}(t) + C_M\{\dot{\boldsymbol{x}}(t) \times \boldsymbol{\omega}(\boldsymbol{t})\} \quad (\mathbf{3})$$

Where $\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), \ddot{\boldsymbol{x}}(t)$ represent the ball's position [m], velocity [m/s] and acceleration [m/s²], at time t in the global reference frame. $\boldsymbol{\omega}$ represents the ball's spin [rad/s], m the mass of the ball [g], with $m\ddot{\boldsymbol{x}}_{\boldsymbol{D}}$ and $m\ddot{\boldsymbol{x}}_{\boldsymbol{M}}$ being the drag and Magnus forces, C_D and C_M the drag and Magnus constants.

Equations (4) and (5) describe rebound phenomenon, based on [47]. These equations are valid for a rolling ball without sliding (i.e., with low speed). The parameter values were experimentally obtained in [47] using an ABS ping-pong ball – parameters are different with a celluloid ball. Equations for the sliding phase are also available in [47].

$$\dot{\boldsymbol{x}}' = \boldsymbol{A}_{\boldsymbol{v}}\dot{\boldsymbol{x}} + \boldsymbol{B}_{\boldsymbol{v}}\boldsymbol{\omega} \quad (\mathbf{4})$$

$$\boldsymbol{\omega}' = \boldsymbol{A}_{\boldsymbol{w}}\dot{\boldsymbol{x}} + \boldsymbol{B}_{\boldsymbol{w}}\boldsymbol{\omega} \quad (\mathbf{5})$$

With

$$\boldsymbol{A}_{\boldsymbol{v}} = \begin{bmatrix} \frac{3}{5} & 0 & 0 \\ 0 & \frac{3}{5} & 0 \\ 0 & 0 & -C_{RZ} \end{bmatrix}, \boldsymbol{B}_{\boldsymbol{v}} = \begin{bmatrix} 0 & -\frac{2r}{5} & 0 \\ \frac{2r}{5} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \boldsymbol{A}_{\boldsymbol{w}} = \begin{bmatrix} 0 & \frac{3}{5r} & 0 \\ \frac{-3}{5r} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \boldsymbol{B}_{\boldsymbol{w}} = \begin{bmatrix} 2/5 & 0 & 0 \\ 0 & 2/5 & 0 \\ 0 & 0 & C_{\omega z} \end{bmatrix}$$

$$C_{\omega z} = 1 - \frac{3\tau cb}{2r^2}\frac{\|\vec{v}^P\|}{\|\vec{v}^M\|}|\dot{x}_z|^{0.75}u$$

With $u = \mu(1 + C_{rz})\frac{|\dot{x}_z|}{\|\vec{v}^P\|}$, $\|\vec{v}^M\| = \sqrt{(\omega_z\tau)^2 + a^2}$. Where $\mu = 0.22$ is the dynamic coefficient of friction between the ball and the table, $C_{rz} = 0.9$ is the coefficient of restitution of the table in the z direction, $\tau = 0.5\,ms$ is the duration of the ball/table contact, $r = 0.02\,m$ is the radius of the ball, and $\vec{v}^P$is the tangential velocity defined by:

$$\vec{v}^P = \begin{bmatrix} \dot{x}_x - r\omega_y \\ \dot{x}_y - r\omega_x \\ 0 \end{bmatrix}$$

To evaluate the prediction accuracy in simulation, we generated 2000 random trajectories with random initial conditions – position, translational and rotational velocities. Trajectories were generated using the model described above, lasting for 1 second at 200 fps. Out of these, 1800 trajectories were used for training the data-driven models, while the remaining 200 were reserved for testing.

For testing accuracy under noisy conditions, Gaussian noise with a maximum amplitude of 10 mm was added to the simulated dataset, as follow:

$$\psi = N(0; 0.002) \tag{6}$$

Where $\psi \in [-0.01; 0.01]$. The maximum amplitude and variance were chosen to reflect the accuracy of the real vision system. Second, for real ball trajectories, three infrared Optitrack cameras (100 fps) and a ping-pong ball covered in infrared tape were used. Over 300 real trajectories, including topspin and backspin shots with table rebounds, were captured to ensure sufficient data for training.

4.2.4 Evaluation strategy

The accuracy of the methods was evaluated under three configurations. First, the methods were trained and tested on a set of perfect simulated data. Secondly, to assess their performance under more realistic conditions, Gaussian noise was added with a maximum of 1 centimeter. Finally, all methods were tested on the real dataset to evaluate their usability and accuracy in real-world conditions.

For the accuracy evaluation on the simulated dataset, predictions were performed over 130 positions at 200 fps, using 30 to 100 observations. For the real dataset, predictions were made on 90 positions at 100 fps, using 15 to 50 observations. To evaluate the performance of each architecture for trajectory prediction, both in simulations and real-world conditions, the following strategy was employed:

Step 1: Dataset Preparation

The training data was generated from either the simulated or real trajectory datasets. In particular, 1800 simulated and more than 250 real-world trajectories were used for training. Two conditions were applied for the simulated data: one with the generated trajectories, and the other with added Gaussian noise (up to 1 centimeter) to simulate realistic, noisy observations. To create input data for recursive architectures, 11 consecutive positions were randomly selected from a dataset. The first 10 positions were added to the input training matrix, and the 11th to the target matrix. This process was repeated to generate a sufficient dataset, with 10,000 samples determined experimentally as optimal for training.

For TAE-based architectures, we followed the approach outlined in [12]. The input data represented a trajectory over 130 (or 90 for real trajectories) time steps. A random number m < 130 (or 90) of points was selected from a random trajectory in the dataset. The resulting input data was a 130x4 (90x4) matrix, where the first m positions were known and the rest zero-padded (see Figure 2). The corresponding output was a 130x3 (90x3) matrix representing the complete trajectory. This procedure was repeated to generate a training dataset with varied numbers of known positions for prediction. Additionally, for the real dataset with missing observations, second-order polynomial interpolation was applied using the df.interpolate function from the Panda library [48].

Step 2: Training the neural networks

The neural networks were trained using Keras [43], employing the Adam optimizer and the Mean Squared Error loss function keras.losses.MeanSquaredError. Specific training parameters such as learning rate, number of epochs, and batch size were tuned for each method. Detailed results and training parameters are provided in the Appendix.

Step 3: Accuracy testing

The trained neural networks were evaluated on a test dataset comprising 10% of the trajectories that were not used during training. The architectures were tested by predicting trajectories based on 30 to 100 observations for the simulated data, and 15 to 30 observations for the real dataset, corresponding to observation times of 0.15 s to 0.5 s.

Step 4: Error analysis

To assess the overall accuracy, the root-mean-square error (RMSE) was calculated for all predicted trajectories in the test dataset across all observation intervals. The RMSE was computed as follows: $e = \sqrt{mean(\sum(\boldsymbol{x} - \boldsymbol{x_p})}$, where $\boldsymbol{x}$ represents the true trajectory and $\boldsymbol{x_p}$ the predicted trajectory. The advantage of using RMSE as a metric is that it provides a quick and easily interpretable overview of the accuracy of all tested methods.

For additional insights, RMSE values were also computed based on the number of observations used for prediction, providing detailed results on the degradation of accuracy over time.

The number of trajectories in the test datasets, both simulated and real-world was chosen to ensure a significant average for RMSE calculation. Specifically, this number matches the test dataset

configuration in [12], with 200 simulated trajectories and 40 real-world trajectories (compared to 35 in [12]).\\

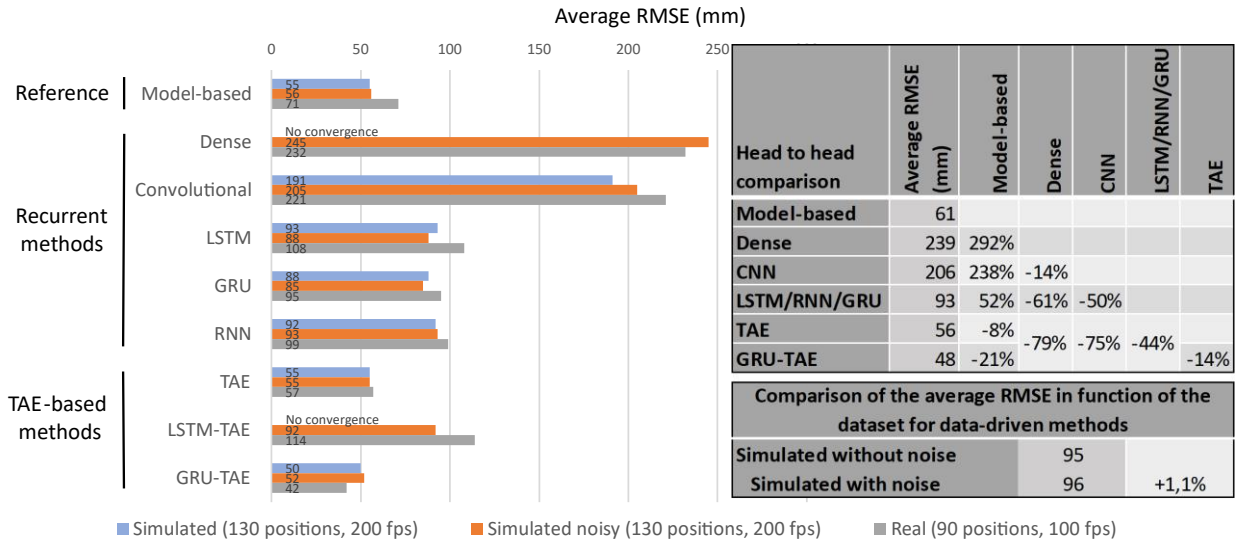
Furthermore, the norm of the position error at the ball's rebound was analyzed as a function of the time elapsed between the last observation and the rebound. The average error at the rebound points and hit points is provided in the Appendix. The hit point is defined as the first point after the ball's rebound, where the ball is at least 10 cm above the table $z > 0.1\ m.$

4.3 Results

4.3.1 Global accuracy

Figure 3 summarizes the average accuracy of the prediction methods by comparing the average RMSE across all predicted positions for 200 test trajectories on three datasets: a perfect simulated dataset (130 positions at 200 fps), a noisy simulated dataset, and a dataset of real-world trajectories (90 positions at 100 fps).

First, Figure 3 shows that the reference model-based prediction achieves an average RMSE of 56 mm and 55 mm when using the simulated dataset with and without noise respectively. For the real-world trajectory dataset, the average RMSE is 71 mm. Second, the dense NN's struggle to converge when using the perfect simulated data, resulting in RMSE of 245 mm and 232 mm for the noisy and real datasets, respectively.



Head to head comparison	Average RMSE (mm)	Model-based	Dense	CNN	LSTM/RNN/GRU	TAE
Model-based	61					
Dense	239	292%				
CNN	206	238%	-14%			
LSTM/RNN/GRU	93	52%	-61%	-50%		
TAE	56	-8%	-79%	-75%	-44%	
GRU-TAE	48	-21%				-14%

Comparison of the average RMSE in function of the dataset for data-driven methods		
Simulated without noise	95	
Simulated with noise	96	+1,1%

Figure 4.3 : Comparison of RMSE for trajectory prediction for simulated and real data.

Third, CNN's yield errors between 191 mm and 221 mm across all datasets, representing an average 14% reduction in RMSE compared to the dense NNs. Recurrent methods using RNN, LSTM and GRU layers have RMSE ranging from 85 mm to 108 mm, which represents a 61% reduction compared to the dense NN.

Fourth, the TAE and GRU-TAE methods achieve an average RMSE reduction of 44% compared to RNN, LSTM and GRU models. In particular, the TAE achieves an RMSE between of 55 mm and 57 mm across all configurations. The LSTM-TAE, however, fails to converge for the simulated dataset, producing RMSEs between 91 mm and 114 mm. Finally, the GRU-TAE achieves an RMSE between 50 mm and 52 mm on the simulated datasets and an RMSE of 42 mm on the real-world dataset, representing an average 14% improvement compared to the TAE. The RMSE of GRU-TAE on real-world data is also 41% smaller than the RMSE of the reference model-based method.

For data-driven methods that do not face convergence issues (i.e., Dense NN and LSTM-TAE), the average RMSE for prediction is 95 mm with perfect simulated data and 96 mm with noisy simulated data, representing only a 1.1% degradation. No comparison is made with real-world data, as the number of predicted positions and the fps differ.

4.3.2 Simulation without noise

Figure 4 compares the accuracy of the main families of architectures as a function of the number of observations predicted before the ball rebounds. Figure 4 indicates that recurrent architectures achieve a lower RMSE than TAE-based methods during the first six prediction steps. However, after 30 predicted observations, TAE-based architectures attain an RMSE of 50 mm, while recurrent architectures exhibit an RMSE between 132 mm and 147 mm. The RMSE for the CNN increases to 416 mm. Additionally, the model-based prediction achieves an RMSE of less than 16 mm during the first 30 steps predicted by the algorithm before the rebound.

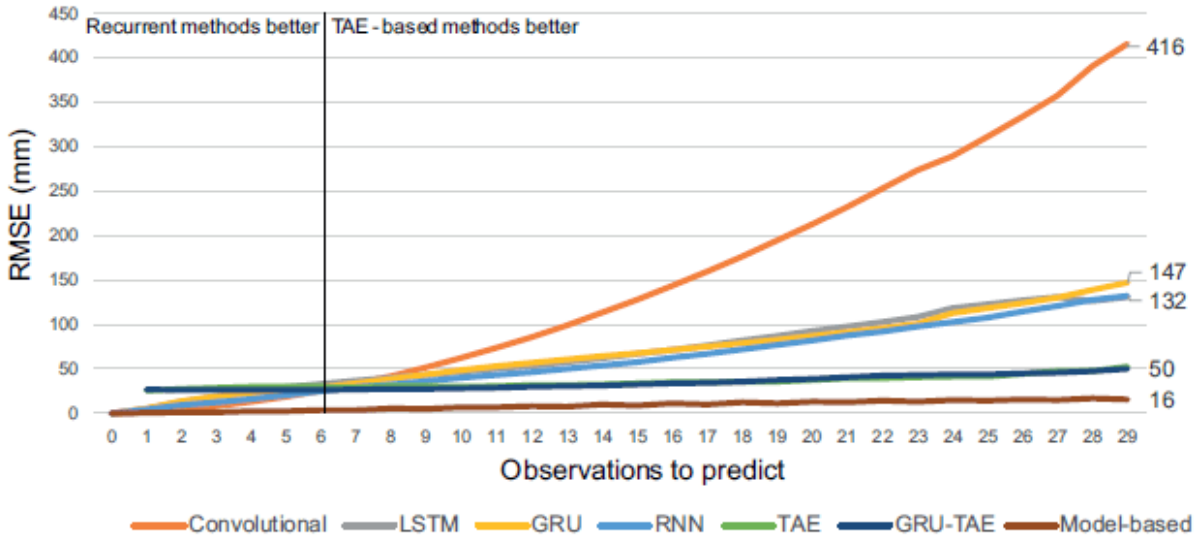


Figure 4.4 : RMSE comparison depending on the number of observations to predict for prefect simulated data, before rebound.

4.3.3 Simulation, with noise

Figure 5 compares the same accuracy for noisy simulated data. Once again, it shows that recurrent architectures achieve a lower RMSE than TAE-based architectures during the first six prediction steps; however, this trend reverses for longer predictions. Specifically, for 30 steps of prediction, the CNN exhibit an RMSE of 447 mm, while the three recurrent methods have RMSE ranging

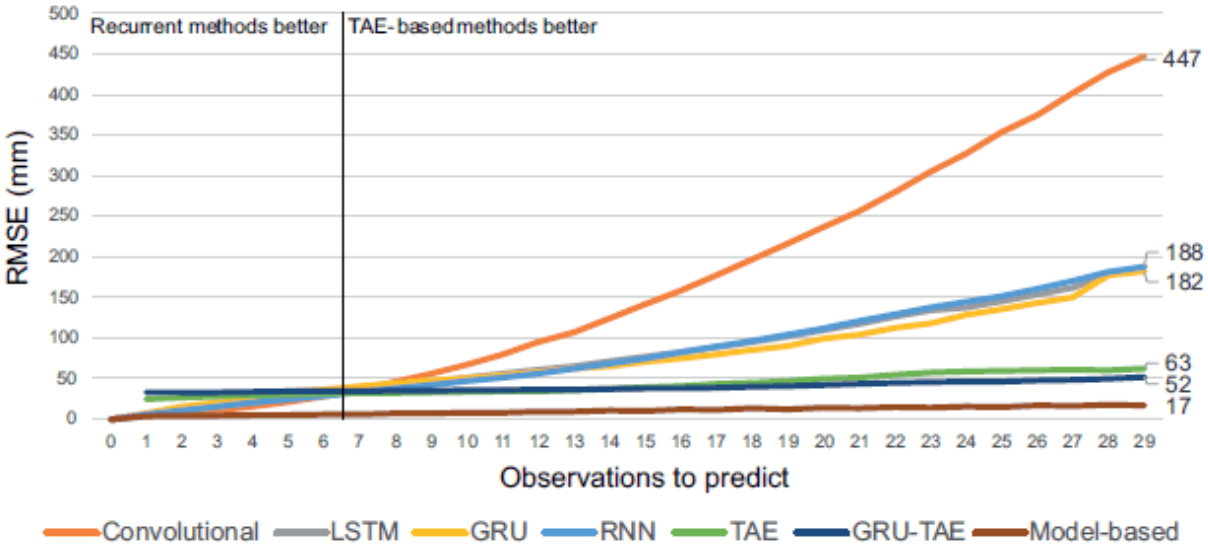


Figure 4.5 : RMSE comparison depending on the number of observations to predict for noisy simulated data, before rebound.

from 182 mm to 188 mm. Contrarily to Figure 4, RMSE of the two TAE-based methods-traditional TAE and GRU-TAE differ and are 63 mm and 52 mm respectively. The reference model-based prediction continues to achieve an RMSE of less than 17 mm when fewer than 30 observations are predicted by the algorithm, before the rebound.

4.3.4 Real dataset

Figure 6 illutrates the degradation in prediction accuracy for the real dataset. First, recurrent methods maintain a lower RMSE than TAE-based methods for the first 11 predicted observations. For 30 predicted observations, the CNN achieves an RMSE of 403 mm, while LSTM NNs show an RMSE of 232 mm, compared to 169 mm for the GRU and RNN models. As before, there is a

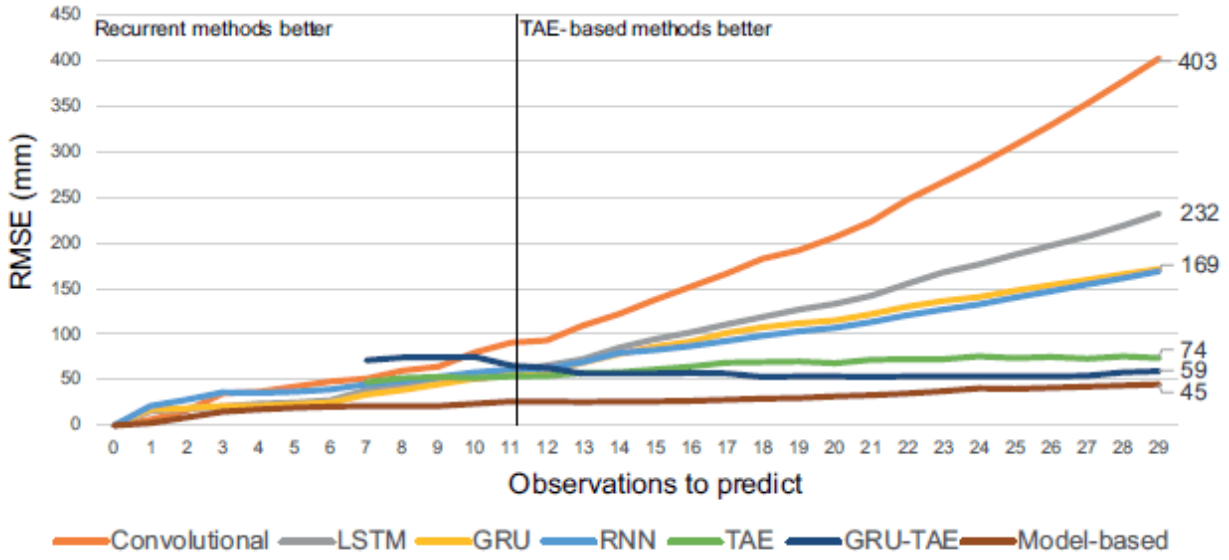


Figure 4.6 : Comparison of RMSE depending on the number of observations to predict for real data, before the rebound.

difference between the TAE and GRU-TAE methods, with RMSE of 74 mm and 59 mm, respectively. Secondly, the reference model-based architecture shows an increasing RMSE as predictions extend further, reaching 45 mm for a 30-step prediction.

Figures 4 to 6 also indicate that for a 0.3 s prediction, TAE-based methods achieve an average RMSE of 58 mm, compared to 171 mm for recurrent methods, representing a 66% improvement in accuracy. Additionally, the GRU-TAE achieves an average RMSE of 54 mm compared to 62 mm for the TAE, resulting in a 13% improvement.

Figure 7 illustrates an example of trajectory prediction using a CNN, GRU, TAE, GRU-TAE and the reference model-based architecture, based on 30 observations of noisy data.

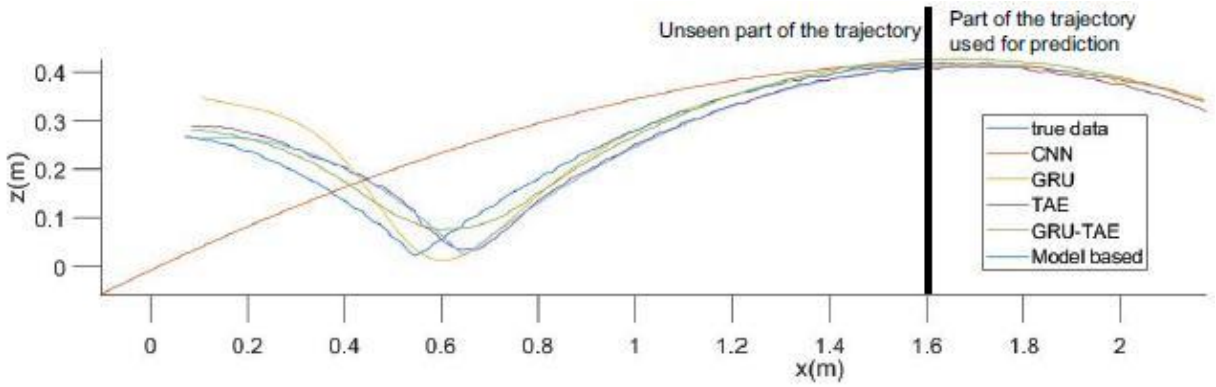


Figure 4.7 : Examples of prediction of a trajectory using different tested methods.

Finally, Table 2 presents a comparison of the time required for training and for predicting a single trajectory. TAE-based methods take between 2 ms and 8.2 ms to compute a trajectory prediction, while recurrent methods require significantly longer, ranging from 160 ms to 220 ms. In contrast, the training time for recurrent methods ranges from 55 s for Dense NNs to 63 s to 148 s for LSTM, GRU, and RNN models, whereas the training time for the tested TAE models varies between 140 s and 289 s.

Tableau 4.2 : Comparison of training and computation time of tested methods

Architecture	**Time to predict one trajectory for 0.65 s**	**Time required for training**
Dense	160 ms	55 s
LSTM, GRU, RNN	220 ms	63 s to 148 s
TAE	2 ms	140 s to 289 s
GRU-TAE	8.2 ms	Around 5000 s
Model-based	1.8 ms	No training

4.4 Discussion

The first main result is about the average accuracy of the compared methods, as shown in Figure 3. It highlights that the TAE-based architectures outperform recurrent approaches, achieving an average reduction of 44% in RMSE for trajectory predictions. This finding aligns with the

comparison of accuracy between the LSTM and the TVAE presented in [12], even though no specific RMSE values were provided. Notably**, the GRU-TAE is the most accurate architecture, surpassing the traditional TAE, with a relative improvement of 14%.** Moreover, the LSTM-TAE encountered convergence issues, particularly when using the perfect simulated dataset. These results will enhance robotic arm performance in returning balls to human players, enabling the system to move to the hitting point earlier and avoiding last-minute adjustments in hitting conditions.

The second main result, illustrated by Figure 3 indicates that **the reference model-based algorithm is less accurate than the TAE-based architectures.** Its RMSE is 8% and 21% bigger compared to TAE and GRU-TAE, respectively. In particular, while the accuracy of the model-based method is comparable to that of TAE and GRU-TAE for simulated data - with RMSE between 50 and 56mm -, there is a significant advantage for GRU-TAE in real-world trajectory predictions, yielding an RMSE of 42 mm compared to 71 mm for the model-based method. The difference between simulated and real-world trajectories can be attributed to inaccuracies in estimating certain parameters of the dynamic model of the real ball, whereas the model used in simulation is ideal. Additionally, the tape fixed on the ball alters its dynamic model. In [12], the difference in accuracy between the TVAE and the model-based approach was even more pronounced for real-world trajectory predictions, largely due to the significant approximations in the dynamic model implemented in that study, such as assuming a rebound coefficient of 1 between the table and the ball.

In addition, Figures 4 to 6 demonstrate that the model-based approach outperforms data-driven methods during the first 30 steps of prediction, prior to the rebound. This is because the prediction is based on differential equations from the dynamic model, resulting in greater accuracy, particularly before the rebound. However, the degradation of the accuracy after the rebound is more important for the model-based method, as spin is not considered, leading to a higher average RMSE. In contrast, the TAE-based architectures implicitly account for spin, based on its impact on the ball’s trajectory. Also, since the dynamic model for real-world trajectories is not perfect, the degradation in accuracy is more important than in simulation, with an average error of 45mm after 30 observations, compared to 16 and 17mm in simulation, respectively.

The third main result illustrated by Figures 4 to 6 is about the understanding of the degradation of predictions as we predict the ball trajectory further into the future. First, these figures indicate that while recurrent methods generally exhibit a higher overall RMSE than TAE-based methods, their predictions are more accurate during the initial steps of prediction. For simulated data, recurrent methods perform better for the first six predicted observations, while this extends to 11 predicted observations when methods were tested on the real dataset. This finding is again consistent with results presented in [12], where the LSTM demonstrated better performances during the first 10 steps of prediction before being outperformed by the TVAE. Additionally, in [14], the seq2seq architecture – GRU-ED – also showed greater error compared to LSTM when predicting fewer than 30 observations. Secondly, another finding consistent with [12] and [14] is that Figures 4 to 6 reveal that all recurrent methods suffer from a cumulative error as predictions extend further into the future. This cumulative error is nearly linear for RNN, LSTM and GRU, leading to an average error between 132 mm and 232 mm for a 30-step prediction. For CNN, the cumulative error is exponential, resulting in poorer accuracy, with an average error between 403 mm and 447 mm for a 30-step prediction—approximately three times greater than that of other recurrent methods.

In addition, it is noteworthy that TAE-based architectures exhibit a static error from the first prediction. This static error is approximately 3 cm for simulated data and 5 cm for real data, which is similar in [12] and [14]. However, unlike recurrent methods, TAE architectures do not suffer from cumulative errors, demonstrating, on average, 66% greater accuracy for a 30-step prediction compared to recurrent methods. Finally, Figures 5 and 6 show that for trajectory predictions from noisy and real observations, replacing dense layers by GRU layers in the encoder results in more accurate long-term predictions, yielding an average relative improvement of 13%. In [14] the use of GRU-ED also reduced prediction errors compared to the TVAE in [12], with RMSE values of 27 mm and 52 mm, respectively.

The fourth main result, illustrated by Figure 3, is that the accuracy of all tested methods does not degrade significantly with noisy observations. There is only a 1.1% average degradation observed when using noisy data. This confirms the robustness of these AI methods. To our knowledge, no study has compared errors from perfect and noisy observations under similar conditions.

Figure 3 also indicates that the reference dense neural networks struggle with perfect simulated data due to overfitting. In contrast, both noisy simulated and real data exhibit reduced overfitting,

resulting in similar RMSE values ranging from 232 mm to 245 mm. Second, CNNs show slightly better performance, with a 14% improvement compared to the dense neural networks, although they still are far from the accuracy achieved by other recurrent methods. LSTM, GRU and RNN demonstrate comparable accuracy, with RMSE between 85 mm and 108 mm that represents a 61% improvement compared to the dense NN. This outcome is not surprising, considering that all these architectures are designed specifically for processing sequential data [29]. However, during the training process, LSTM again exhibited more difficulties to converge compared to GRU and RNN, likely due to its higher number of parameters [29].

Figure 7 enhances our understanding of the strengths at weaknesses of the different prediction methods. First, it reveals that all data-driven methods struggle to accurately model the rebound mechanism, resulting in poor prediction accuracy during this phase of the trajectory. The CNN fails to predict the occurence of the rebound. Although other recurrent methods can predict the rebound, the recursive algorithm used for predictions introduces a cumulative error that cannot be corrected. On the other hand, Figure 7 shows that for TAE-based methods, the accuracy improves after the rebound and remains satisfying for the reamining part of the trajectory. In [14], the impact of the rebound on the prediction is not discussed. On the contrary, the model-method can comprehend the rebound phenomenon. However, the spin is neglected, leading to inaccuracies in predicting the trajectory following the rebound [15].

Finally, an interesting result presented by Table 2 is that although training recursive methods is faster than training TAE, particularly the GRU-TAE, the recursive algorithm poses challenges for real-time applications, requiring at least 160 ms to predict a 0.65 s trajectory. On the contrary, since TAE methods take an entire trajectory as input, their training process is longer, but only one step is needed for trajectory prediction. Thus, a 0.65 s prediction can be accomplished in 2 ms to 8.2 ms, allowing for an update with each new position of the ball computed by the camera system operating at 100 Hz. It is important to note that the computation time for TAE-based methods is longer than the 1.8 ms required by the model-based prediction algorithm. However, since the computation time for TAE-based architectures is shorter than the camera frequency, this does not hinder real-time use. In particular, unlike to recursive and model-based methods, TAE-based architectures do not experience delays when predicting further trajectories. In [12], similar computation times are obtained; the prediction using the TVAE ranges from 8 ms and 10 ms, consistent with the calculation time of our TAE and GRU-TAE. In [14], the GRU-ED takes

between 5 ms and 6 ms for predictions, which is 10 times longer than the time required for curve fitting or a model-based predictions.

The main limitation of this study is that model-based methods incorporating spin estimation were not considered. In fact, the state-of-the-art indicates that these methods are highly sensitive to the accuracy of dynamic parameters. Future works could explore parameter optimization using machine learning techniques to integrate spin estimation with model-based methods. In addition, results indicate that each approach performs optimally at a different stage of the prediction: recursive architectures excel in the initial steps, model-based methods effectively interpret ball bounces and TAE-based architectures are more efficient for the remaining part of the trajectory. Therefore, future works could combine these approaches to enhance overall prediction accuracy.

Finally, these prediction methods are part of the algorithm for robotic table tennis. Results demonstrate the usability of TAE and GRU-TAE for real-time applications. Future works will integrate these methods to improve the performance of physical table tennis robots.

4.5 Conclusion

The objective of this paper was to use a GRU-TAE to predict trajectories of a ping-pong ball. Specifically, the study compared the accuracy of GRU-TAE with more traditional time-series prediction machine learning methods for ping-pong ball trajectory predictions. Results were demonstrated using both real and simulated ping-pong ball trajectory datasets. The main results were as follows: 1. TAE-based architectures outperform recurrent approaches, reducing RMSE in trajectory prediction by 44%, and the reference model-based method by 41% for real-world data. In particular, GRU-TAE outperform traditional TAE, with a relative improvement of 14%. 2. There is only a 1.1% average degradation in performance for all methods when using the noisy dataset compared to the perfect one, demonstrating the robustness of the tested AI methods to noisy observations. 3. GRU-TAE can be used for real-time predictions, achieving an average calculation time of 8.2 ms for each trajectory prediction. 4. Recurrent methods are more accurate than TAE-based methods for short-term predictions, while TAE-based methods perform well for longer-term predictions. These results are important because they enhance robotic arm performance in returning balls to human players, allowing earlier positioning for ball return, and reducing late position adjustments. Future work could combine TAE-based methods with model-based and recurrent methods to optimize prediction accuracy. Perspectives are to compare and integrate these methods

with model-based prediction techniques and apply them to a real-world robotic ping-pong player system.

4.6 Statements and Declarations

Funding: Institute for data valorization (IVADO), Montreal, Canada.

Author contributions: Conceptualization: BT, MR Methodology: BT, MR. Investigation: BT, MR. Visualization: BT, MR. Funding acquisition: BT, MR. Project administration: BT, MR. Supervision: MR. Writing – original draft: BT. Writing – review & editing: BT, MR.

Competing interests: Authors declare that they have no competing interests.

4.7 Appendix

4.7.1 Implementation of the reference architectures

4.7.1.1 Recursive methods

The tested models using the recursive approach included dense networks, RNN, LSTM, GRU and CNN. The inputs of these neural networks were 10x3 matrices representing the last n measured positions of the ball along the x, y and z axes. These inputs were used by the models to predict the next ball position, represented as a 1x3 vector. After predicting the next position, this new prediction could be fed back as input for the following prediction. This recursive process allows for the prediction of the entire trajectory. The general recursive architecture is illustrated in Figure 8.

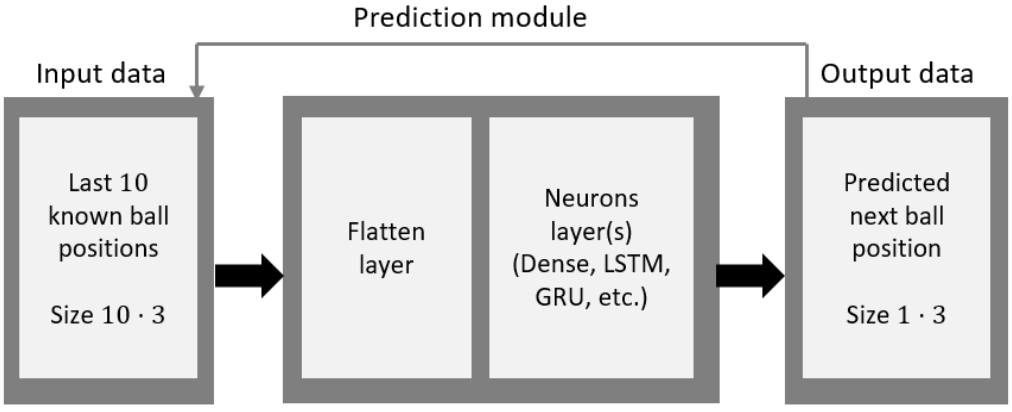


Figure 4.8 : General architecture of recursive methods for ping-pong ball trajectory prediction.

4.7.1.2 TAE-based methods

The tested models using the TAE approach included a simple TAE with dense layers and an LSTM-TAE, where specific dense layers were replaced with LSTM layers. The training and testing

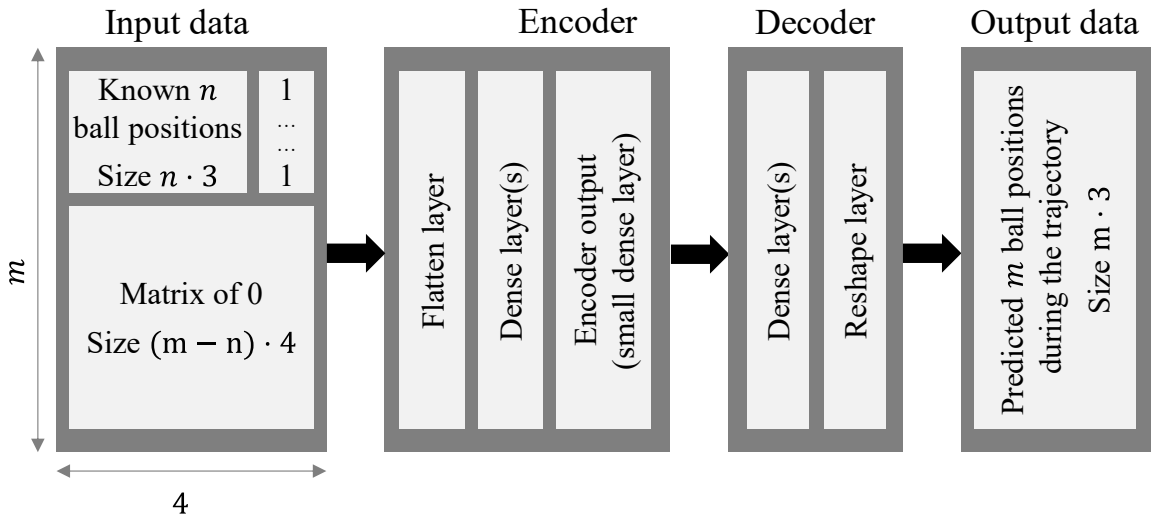


Figure 4.9 : General architecture of a TAE-based neural network for ping-pong ball trajectory prediction.

methodology was inspired by [12]. To predict a trajectory consisting of 130 positions (or 90 for the real dataset), the inputs were a 130x4 (90x4) matrix containing the known n positions of the ball along the x, y and z axes throughout the trajectory, along with a zero-padded mask to fill the matrix. This fixed length was used for all training and test data trajectories. The output of these models was a 130x3 (90x3) matrix containing the x, y and z positions of the ball for the entire predicted trajectory as shown in Figure 9.

4.7.1.3 Model-based architecture

To evaluate the overall accuracy of data-driven methods, a traditional model-based architecture has also been implemented. The baseline method is inspired by [7], and predicts the ball's trajectory without considering spin. This method is based on simplified dynamic equations (1) and (2), as follows:

$$m\ddot{\boldsymbol{x}}(\boldsymbol{t}) = -m\boldsymbol{g} - m\ddot{\boldsymbol{x}}_{\boldsymbol{D}}(t) = -m\boldsymbol{g} - C_D\|\dot{\boldsymbol{x}}(t)\|\dot{\boldsymbol{x}}(t) \qquad (\mathbf{1})$$

Where $x(t), \dot{x}(t)$ and $\ddot{x}(t)$ are the ball's position [m], velocity [m/s] and acceleration [m/s²], at time t in the global reference frame, respectively. The term ω represents the ball's spin [rad/s], m is the ball's mass [g], $m\ddot{x}_D$ the drag force, C_D [g/m] the drag constant.

$$\dot{\boldsymbol{x}}' = \boldsymbol{A_v}\dot{\boldsymbol{x}} \tag{2}$$

Where

$$\boldsymbol{A_v} = \begin{bmatrix} \frac{3}{5} & 0 & 0 \\ 0 & \frac{3}{5} & 0 \\ 0 & 0 & -C_{RZ} \end{bmatrix}$$

Crz = 0.9 is the coefficient of restitution of the table in the z-direction.

Using these equations, the initial position is defined as the first detected point of the trajectory, and the initial velocity is iteratively computed using the first and last measured points before the rebound. The ball's spin is neglected and set to 0.

4.7.2 Detailed results

Method	Approach	Layer(s)	Learning Rate	Batch size	Number of epochs	Number of trajectories for training	Number of data for training	Length of prediction	Shift	Dataset (1=simulated, 2=noisy simulated, 3=real)	Additionnal notes	Prediction = input + output (delta)	Average RMSE	Average error at rebound point	Average error at hit point	RMSE after 0,01s	RMSE after 0,05s	RMSE after 0,1s	RMSE after 0,2s	RMSE after 0,29s
LSTM	Recursive	L4	0.01	4096	500	1800	10000	10	1	1		O	Not computed	0.076	0.094	0.025	0.040	0.058	0.100	0.128
LSTM	Recursive	L8	0.01	4096	500	1800	10000	10	1	1		O	Not computed	0.113	0.097	0.014	0.044	0.063	0.145	0.319
LSTM	Recursive	L16	0.01	4096	500	1800	10000	10	1	1		O	Not computed	0.061	0.098	0.015	0.029	0.045	0.085	0.030
LSTM	Recursive	L32	0.01	4096	500	1800	10000	10	1	1		O	0.128	0.063	0.083	0.013	0.031	0.037	0.082	0.061
LSTM	Recursive	L64	0.01	4096	500	1800	10000	10	1	1		O	0.097	0.069	0.076	0.018	0.042	0.052	0.089	0.066
LSTM	Recursive	L128	0.01	4096	500	1800	10000	10	1	1		O	0.081	0.079	0.118	0.011	0.031	0.054	0.110	0.159
LSTM	Recursive	L4L4	0.01	4096	500	1800	10000	10	1	1		O	Not computed	0.081	0.089	0.014	0.041	0.064	0.116	0.055
LSTM	Recursive	L8L8	0.01	4096	500	1800	10000	10	1	1		O	Not computed	0.072	0.112	0.009	0.036	0.048	0.088	0.165
LSTM	Recursive	L16L16	0.01	4096	500	1800	10000	10	1	1		O	0.085	0.074	0.085	0.013	0.027	0.048	0.108	0.050
LSTM	Recursive	L32L32	0.01	4096	500	1800	10000	10	1	1		O	Not computed	0.090	0.085	0.008	0.021	0.049	0.138	0.102
LSTM	Recursive	L64L64	0.01	4096	500	1800	10000	10	1	1		O	0.074	0.070	0.087	0.009	0.035	0.056	0.098	0.057
LSTM	Recursive	L4D4	0.01	4096	500	1800	10000	10	1	1		O	Not computed	0.079	0.119	0.011	0.052	0.071	0.100	0.031
LSTM	Recursive	L8D8	0.01	4096	500	1800	10000	10	1	1		O	Not computed	0.080	0.097	0.014	0.032	0.051	0.110	0.081
LSTM	Recursive	L16D16	0.01	4096	500	1800	10000	10	1	1		O	Not computed	0.066	0.111	0.011	0.028	0.054	0.089	0.079
LSTM	Recursive	L32D32	0.01	4096	500	1800	10000	10	1	1		O	Not computed	0.086	0.074	0.018	0.044	0.056	0.117	0.064
LSTM	Recursive	L4	0.01	4096	500	1800	10000	10	1	1		X	Not computed	0.409	0.420	0.069	0.235	0.402	0.500	0.872
LSTM	Recursive	L16	0.01	4096	500	1800	10000	10	1	1		X	Not computed	0.247	0.209	0.027	0.142	0.245	0.311	0.310
LSTM	Recursive	L16	0.01	4096	500	1800	10000	10	1	1	speed in	O	Not computed	0.460	0.542	0.035	0.190	0.357	0.635	0.776
LSTM	Recursive	L16L16	0.01	4096	500	1800	10000	10	1	1	inputs	O	Not computed	0.502	0.584	0.040	0.203	0.383	0.694	0.863
LSTM	Recursive	L16	0.01	4096	500	1800	100000	10	1	1		O	Not computed	0.081	0.084	0.007	0.033	0.049	0.116	0.215
LSTM	Recursive	L16	0.01	4096	500	1800	5000	10	1	1		O	Not computed	0.077	0.088	0.007	0.045	0.054	0.112	0.068
LSTM	Recursive	L16	0.01	4096	500	1800	2000	10	1	1		O	Not computed	0.091	0.130	0.010	0.052	0.073	0.126	0.107
LSTM	Recursive	L16	0.01	4096	500	1800	10000	10	10	1		O	Not computed	0.541	0.628	0.040	0.203	0.410	0.743	0.937
LSTM	Recursive	L16	0.01	4096	500	1800	10000	20	1	1		O	Not computed	0.070	0.100	0.011	0.025	0.047	0.099	0.125
LSTM	Recursive	L16	0.01	4096	500	1800	10000	30	1	1		O	Not computed	0.061	0.094	0.010	0.024	0.044	0.085	0.089
LSTM	Recursive	L8	0.01	4096	500	1800	10000	30	1	1		O	Not computed	0.086	0.074	0.016	0.030	0.046	0.122	0.114
LSTM	Recursive	L32	0.01	4096	500	1800	10000	30	1	1		O	Not computed	0.089	0.086	0.007	0.025	0.052	0.135	0.094
LSTM	Recursive	L4	0.01	4096	500	1800	10000	30	1	1		O	Not computed	0.071	0.083	0.008	0.027	0.051	0.105	0.036
LSTM	Recursive	L16L16	0.01	4096	500	1800	10000	30	1	1		O	Not computed	0.067	0.092	0.006	0.018	0.041	0.091	0.147
LSTM	Recursive	L32L32	0.01	4096	500	1800	10000	30	1	1		O	Not computed	0.140	0.099	0.009	0.039	0.091	0.199	0.246
LSTM	Recursive	L4L4	0.01	4096	500	1800	10000	30	1	1		O	Not computed	0.069	0.083	0.008	0.024	0.052	0.103	0.057
LSTM	Recursive	L4D4	0.01	4096	500	1800	10000	30	1	1		O	Not computed	0.070	0.079	0.018	0.033	0.043	0.080	0.147
LSTM	Recursive	D64	0.01	4096	500	1800	10000	1	10	1		O	Not computed	0.444	0.517	0.050	0.113	0.298	0.628	0.819
LSTM	Recursive	D64D64	0.01	4096	500	1800	10000	1	10	1		O	Not computed	0.418	0.491	0.050	0.102	0.277	0.589	0.778
LSTM	Recursive	D8	0.01	4096	500	1800	10000	1	10	1		O	Not computed	0.449	0.509	0.050	0.121	0.307	0.635	0.850
LSTM	Recursive	L8	0.01	4096	500	1800	10000	1	10	1		O	Not computed	0.321	0.321	0.036	0.100	0.209	0.414	0.578
LSTM	Recursive	L64	0.01	4096	500	1800	10000	1	10	1		O	Not computed	0.334	0.364	0.045	0.083	0.222	0.454	0.601

Method	Approach	Layer(s)	Learning Rate	Batch size	Number of epochs	Number of trajectories for training	Number of data for training	Length of prediction	Shift	Dataset (1=simulated, 2=noisy simulated, 3=real)	Additionnal notes	Prediction = input + output (delta)	Average RMSE	Average error at rebound point	Average error at hit point	RMSE after 0,01s	RMSE after 0,05s	RMSE after 0,1s	RMSE after 0,2s	RMSE after 0,29s
GRU	Recursive	G4	0.0100	4096	500	1800	10000	10	1	1		O	Not computed	0.087	0.112	0.013	0.033	0.080	0.113	0.167
GRU	Recursive	G8	0.0100	4096	500	1800	10000	10	1	1		O	Not computed	0.084	0.093	0.015	0.028	0.039	0.095	0.315
GRU	Recursive	G16	0.0100	4096	500	1800	10000	10	1	1		O	0.080	0.056	0.075	0.018	0.030	0.042	0.074	0.062
GRU	Recursive	G32	0.0100	4096	500	1800	10000	10	1	1		O	Not computed	0.074	0.096	0.010	0.034	0.064	0.094	0.078
GRU	Recursive	G4G4	0.0100	4096	500	1800	10000	10	1	1		O	Not computed	0.181	0.233	0.018	0.049	0.069	0.122	0.984
GRU	Recursive	G16G16	0.0100	4096	500	1800	10000	10	1	1		O	0.086	0.063	0.087	0.014	0.021	0.039	0.080	0.098
Conv	Recursive	C16	0.0050	4096	500	1800	10000	10	1	1		O	Not computed	0.140	0.153	0.004	0.022	0.063	0.197	0.375
Conv	Recursive	C4	0.0020	4096	500	1800	10000	10	1	1		O	Not computed	0.210	0.226	0.005	0.029	0.087	0.296	0.527
Conv	Recursive	C8	0.0020	4096	500	1800	10000	10	1	1		O	0.200	0.136	0.130	0.004	0.024	0.067	0.197	0.315
Conv	Recursive	C32	0.0020	4096	500	1800	10000	10	1	1		O	0.194	0.168	0.179	0.005	0.027	0.078	0.241	0.404
Dense	Recursive	D16	0.0020	4096	500	1800	10000	10	1	1		O	Not computed	0.909	1.538	0.026	0.070	0.238	1.142	6.023
GRU	Recursive	G16	0.0100	4096	500	1800	10000	10	1	2		O	Not computed	0.066	0.097	0.008	0.024	0.042	0.092	0.087
LSTM-TAE	Seq2seq	D50B16L16	0.0005	512	100	1800	10000	130		2			Not computed	0.146	0.108	0.126	0.133	0.136	0.150	0.159
LSTM-TAE	Seq2seq	D30B16L8	0.0005	512	100	1800	10000	130		2			Not computed	0.136	0.118	0.115	0.115	0.126	0.140	0.146
LSTM-TAE	Seq2seq	D30B16L8	0.0010	512	200	1800	10000	130		2			Not computed	0.130	0.086	0.117	0.123	0.127	0.133	0.142
LSTM-TAE	Seq2seq	D16B16L4	0.0010	512	200	1800	10000	130		2			Not computed	0.206	0.144	0.202	0.203	0.206	0.209	0.200
LSTM-TAE	Seq2seq	D100B16L4	0.0010	512	200	1800	10000	130		2			Not computed	0.193	0.125	0.183	0.187	0.191	0.195	0.197
LSTM-TAE	Seq2seq	D30B16L16	0.0010	512	200	1800	10000	130		2			Not computed	0.086	0.072	0.059	0.064	0.075	0.099	0.098
LSTM-TAE	Seq2seq	D30B16L32	0.0010	512	200	1800	10000	130		2			Not computed	0.087	0.077	0.065	0.071	0.076	0.100	0.098
LSTM-TAE	Seq2seq	D30B16L32	0.0010	16	200	1800	10000	130		2			Not computed	0.096	0.083	0.066	0.077	0.090	0.108	0.105
LSTM-TAE	Seq2seq	D30B16L128	0.0010	16	200	1800	10000	130		1			Not computed	0.056	0.065	0.035	0.041	0.047	0.060	0.089
LSTM-TAE	Seq2seq	D100B16L128	0.0010	16	200	1800	10000	130		1			Not computed	0.074	0.078	0.043	0.052	0.067	0.091	0.085
TAE	Seq2seq	D100B16D200-300	0.0010	16	200	1800	10000	130		1			Not computed	0.046	0.051	0.035	0.040	0.041	0.048	0.063
TAE	Seq2seq	D100B16D200	0.0010	16	300	1800	10000	130		1			Not computed	0.035	0.044	0.022	0.023	0.029	0.038	0.049
TAE	Seq2seq	D32B16D32	0.0010	16	300	1800	10000	130		1			Not computed	0.039	0.047	0.027	0.031	0.032	0.042	0.054
TAE	Seq2seq	D16B16D16	0.0010	16	300	1800	10000	130		1			Not computed	0.043	0.048	0.032	0.036	0.038	0.046	0.052
TAE	Seq2seq	D512B16D512	0.0010	16	300	1800	10000	130		1			Not computed	0.038	0.049	0.031	0.033	0.032	0.039	0.055
TAE	Seq2seq	D8B16L16	0.0010	16	300	1800	10000	130		1			Not computed	0.100	0.090	0.070	0.078	0.087	0.108	0.133
TAE	Seq2seq	D100B16D100	0.0010	256	200	1800	10000	130		1	drop 0,2		Not computed	0.066	0.056	0.050	0.054	0.058	0.071	0.088
TAE	Seq2seq	D400B16D400	0.0010	256	200	1800	10000	130		1	drop 0,5		Not computed	0.073	0.062	0.057	0.062	0.065	0.080	0.091
TAE	Seq2seq	D32B16D32	0.0010	256	200	1800	10000	130		1	drop 0,3		Not computed	0.085	0.073	0.071	0.076	0.082	0.092	0.099
TAE	Seq2seq	D200B16D200	0.0010	256	200	1800	10000	130		2	drop 0,2		Not computed	0.069	0.056	0.051	0.056	0.063	0.074	0.089
TAE	Seq2seq	D200B16D200	0.0010	256	200	1800	10000	130		2	drop 0,2		Not computed	0.072	0.062	0.055	0.060	0.067	0.076	0.079
TAE	Seq2seq	D200B16D200	0.0010	256	200	1800	10000	130		1			Not computed	0.048	0.045	0.033	0.036	0.041	0.054	0.059
LSTM-TAE	Seq2seq	L32B16L16	0.0004	128	50	1800	10000	130		1			Not computed	0.194	0.142	0.194	0.201	0.202	0.189	0.175
LSTM-TAE	Seq2seq	L128B32L128	0.0001	128	50	1800	10000	130		1			Not computed	0.155	0.136	0.144	0.148	0.151	0.156	0.164
GRU-TAE	Seq2seq	G32G16G32	0.0001	32	25	1800	10000	130		1			Not computed	0.168	0.135	0.158	0.163	0.166	0.170	0.170
GRU-TAE	Seq2seq	G32G16G32	0.0001	32	50	1800	10000	130		1			Not computed	0.141	0.115	0.120	0.129	0.135	0.146	0.155
GRU-TAE	Seq2seq	G32G16G32	0.0003	32	40	1800	10000	130		1			Not computed	0.091	0.082	0.059	0.068	0.080	0.102	0.119
GRU-TAE	Seq2seq	G64G16G64	0.0003	32	80	1800	10000	130		1			Not computed	0.035	0.043	0.021	0.022	0.026	0.042	0.048
GRU-TAE	Seq2seq	G128G16G128	0.0005	64	80	1800	10000	130		1			Not computed	0.062	0.071	0.048	0.051	0.056	0.070	0.069
GRU-TAE	Seq2seq	G32G16G32	0.0005	64	100	1800	10000	130		1			Not computed	0.069	0.060	0.045	0.049	0.057	0.080	0.088
GRU-TAE	Seq2seq	G32G16G32	0.0010	64	100	1800	10000	130		1			Not computed	0.037	0.050	0.026	0.028	0.031	0.041	0.048
GRU-TAE	Seq2seq	G32G16G32	0.0015	64	300	1800	10000	130		1			Not computed	0.031	0.047	0.025	0.026	0.027	0.031	0.041
GRU-TAE	Seq2seq	G64G16G64	0.0015	64	200	1800	10000	130		1			Not computed	0.020	0.031	0.018	0.017	0.018	0.020	0.025
TAE	Seq2seq	D200B16D200	0.0015	64	200	1800	10000	1300		1			Not computed	0.046	0.046	0.026	0.030	0.036	0.056	0.064
TAE	Seq2seq	D200B16D200	0.0015	64	400	1800	10000	1300		1			Not computed	0.034	0.040	0.019	0.021	0.026	0.045	0.043
TAE	Seq2seq	D200B16D200	0.0015	64	1000	1800	10000	1300		1			Not computed	0.032	0.042	0.021	0.022	0.026	0.036	0.043
TAE	Seq2seq	D400B16D400	0.0015	64	1000	1800	10000	1300		1			Not computed	0.039	0.045	0.023	0.024	0.028	0.051	0.044
TAE	Seq2seq	D400B16D400	0.0015	64	1000	1800	10000	1300		1			0.056	0.036	0.045	0.026	0.028	0.031	0.040	0.047
TAE	Seq2seq	D200B16D200	0.0010	512	1000	1800	10000	1300		1			0.053	0.037	0.041	0.028	0.030	0.028	0.038	0.059
TAE	Seq2seq	D100B16D100	0.0010	512	1000	1800	10000	1300		1			0.054	0.036	0.040	0.028	0.031	0.033	0.039	0.049

Method	Approach	Layer(s)	Learning Rate	Batch size	Number of epochs	Number of trajectories for training	Number of data for training	Length of prediction	Shift	Dataset (1=simulated, 2=noisy simulated, 3=real)	Additionnal notes	Prediction = input + output (delta)	Average RMSE	Average error at rebound point	Average error at hit point	RMSE after 0,01s	RMSE after 0,05s	RMSE after 0,1s	RMSE after 0,2s	RMSE after 0,29s
LSTM	Recursive	D64	0.0100	4096	500	1800	10000	10	1	1		O	0.616	0.469	0.616	0.013	0.073	0.205	0.638	1.232
Dense	Recursive	D64	0.0100	4096	500	1800	10000	1	10	1		O	0.432							
Dense	Recursive	D512	0.0100	4096	500	1800	10000	1	10	1		O	0.406							
Dense	Recursive	D512D512	0.0100	4096	500	1800	10000	1	10	1		O	0.401							
GRU-TAE	Seq2seq	G64	0.0010	64	1000	1800	10000	130		1			0.048	0.030	0.032	0.023	0.022	0.024	0.036	0.043
GRU-TAE	Seq2seq	G16	0.0010	512	1000	1800	10000	130		1			0.052	0.041	0.033	0.029	0.030	0.034	0.045	0.058
LSTM	Recursive	L64	0.0100	4096	500	1800	10000	10	1	2		O	0.085	0.067	0.098	0.011	0.017	0.041	0.093	nan
LSTM	Recursive	L128	0.0100	4096	500	1800	10000	10	1	2		O	0.082	0.060	0.103	0.012	0.024	0.053	0.080	nan
LSTM	Recursive	L16L16	0.0100	4096	500	1800	10000	10	1	2		O	0.070	0.071	0.092	0.012	0.029	0.048	0.100	0.102
LSTM	Recursive	L64L64	0.0100	4096	500	1800	10000	10	1	2		O	0.067	0.071	0.075	0.006	0.022	0.053	0.105	0.052
LSTM	Recursive	L32	0.0100	4096	500	1800	10000	10	1	2		O	0.134	0.166	0.125	0.022	0.058	0.087	0.211	0.482
GRU	Recursive	G32	0.0100	4096	500	1800	10000	10	1	2		O	0.068	0.060	0.089	0.012	0.019	0.037	0.086	0.076
GRU	Recursive	G16	0.0100	4096	500	1800	10000	10	1	2		O	0.097	0.127	0.088	0.024	0.059	0.081	0.160	0.228
GRU	Recursive	G16G16	0.0100	4096	500	1800	10000	10	1	2		O	0.089	0.058	0.086	0.011	0.032	0.045	0.066	0.105
Dense	Recursive	D512	0.0100	4096	500	1800	10000	10	1	2		O	1.097							
Dense	Recursive	D512D512	0.0100	4096	500	1800	10000	10	1	2		O	0.145							
Dense	Recursive	D64D64	0.0100	4096	500	1800	10000	10	1	2		O	0.264							
Dense	Recursive	D128D128	0.0100	4096	500	1800	10000	10	1	2		O	0.326							
Conv	Recursive	C128	0.0100	4096	500	1800	10000	10	1	2		O	0.200							
Conv	Recursive	C32	0.0100	4096	500	1800	10000	10	1	2		O	0.200							
TAE	Seq2seq	D400B16D400	0.0010	64	1000	1800	10000	130		2			0.057	0.042	0.048	0.020	0.024	0.031	0.050	0.069
TAE	Seq2seq	D200B16D200	0.0010	512	1000	1800	10000	130		2			0.054							
TAE	Seq2seq	D100B16D100	0.0010	512	1000	1800	10000	130		2			0.053	0.046	0.037	0.034	0.040	0.038	0.052	0.051
GRU-TAE	Seq2seq	G16B16D100	0.0100	4096	500	1800	10000	130		2			0.054	0.046	0.036	0.035	0.041	0.044	0.050	0.055
GRU-TAE	Seq2seq	G32B16D100	0.0100	4096	500	1800	10000	130		2			0.050	0.040	0.032	0.035	0.035	0.036	0.044	0.052
GRU-TAE	Seq2seq	G32B16D100	0.0100	4096	500	1800	10000	130		2			0.053	0.036	0.041	0.028	0.027	0.028	0.040	0.049
GRU-TAE	Seq2seq	G16G16B16D100	0.0100	4096	500	1800	10000	130		2			0.051	0.039	0.035	0.034	0.035	0.036	0.040	0.045
GRU	Recursive	G64G64	0.0050	64	500	1800	1000	10	1	3		O	0.101	0.111	0.132	0.012	0.033	0.069	0.147	0.194
GRU	Recursive	G32	0.0050	64	500	1800	1000	10	1	3		O	0.136	0.123	0.126	0.026	0.023	0.063	0.167	0.240
GRU	Recursive	G16	0.0010	64	500	1800	1000	10	1	3		O	0.087	0.074	0.084	0.020	0.026	0.049	0.100	0.134
GRU	Recursive	G128	0.0050	64	500	1800	1000	10	1	3		O	0.080	0.071	0.083	0.013	0.019	0.037	0.090	0.146
GRU	Recursive	G512	0.0010	64	500	1800	1000	10	1	3		O	0.071	0.081	0.097	0.022	0.023	0.056	0.105	0.143
LSTM	Recursive	L64	0.0050	64	500	1800	1000	10	1	3		O	0.108	0.139	0.151	0.013	0.019	0.054	0.189	0.310
LSTM	Recursive	L128	0.0050	64	500	1800	1000	10	1	3		O	0.070	0.079	0.091	0.024	0.022	0.047	0.106	0.145
LSTM	Recursive	L16	0.0050	64	500	1800	1000	10	1	3		O	0.088	0.110	0.132	0.014	0.035	0.073	0.152	0.201
LSTM	Recursive	L16L16	0.0050	64	500	1800	1000	10	1	3		O	0.165	0.118	0.126	0.021	0.032	0.066	0.120	0.273
Conv	Recursive	C64	0.0005	64	500	1800	1000	10	1	3		O	0.216	0.157	0.149	0.015	0.045	0.081	0.193	0.354
Conv	Recursive	C512	0.0005	64	500	1800	1000	10	1	3		O	0.259	0.223	0.197	0.017	0.053	0.108	0.285	0.504
Conv	Recursive	C16	0.0005	64	500	1800	1000	10	1	3		O	0.188	0.158	0.144	0.014	0.046	0.083	0.194	0.351
Dense	Recursive	D128	0.0005	64	500	1800	1000	10	1	3		O	0.227	0.155	0.185	0.028	0.049	0.099	0.199	0.310
Dense	Recursive	D512	0.0001	64	500	1800	1000	10	1	3		O	0.179	0.110	0.064	0.023	0.063	0.117	0.132	0.146
Dense	Recursive	D512D512	0.00003	64	500	1800	1000	10	1	3		O	0.290	0.088	0.134	0.027	0.055	0.062	0.096	0.167
TAE	Seq2seq	D100D100	0.0010	16	500	1800	4000	90		3			0.066	0.070	0.075	nan	nan	0.055	0.076	0.084
TAE	Seq2seq	D200D200	0.0010	16	500	1800	4000	90		3			0.059	0.072	0.052	nan	nan	0.071	0.076	0.066
TAE	Seq2seq	D400D400	0.0010	16	500	1800	4000	90		3			0.047	0.055	0.051	nan	nan	0.035	0.063	0.071
GRU-TAE	Seq2seq	G16D400	0.0050	16	500	1800	4000	90		3			0.035	0.051	0.029	nan	nan	0.052	0.048	0.058
GRU-TAE	Seq2seq	G32D400	0.0020	16	500	1800	4000	90		3			0.035	0.047	0.032	nan	nan	0.052	0.041	0.050
GRU-TAE	Seq2seq	G16G16D400	0.0010	16	500	1800	4000	90		3			0.064	0.085	0.034	nan	nan	0.098	0.076	0.076
GRU-TAE	Seq2seq	G64D400	0.0010	16	500	1800	4000	90		3			0.034	0.054	0.042	nan	nan	0.060	0.048	0.054
LSTM-TAE	Seq2seq	L16D400	0.0010	128	500	1800	4000	90		3			0.085	0.122	0.062	nan	nan	0.137	0.112	0.113
LSTM-TAE	Seq2seq	L32D400	0.0010	128	500	1800	4000	90		3			0.089	0.097	0.073	nan	nan	0.096	0.092	0.106
LSTM-TAE	Seq2seq	L64D400	0.0002	128	500	1800	4000	90		3			0.167	0.163	0.136	nan	nan	0.143	0.175	0.195

Method	Approach	Layer(s)	Learning Rate	Batch size	Number of epochs	Number of trajectories for training	Number of data for training	Length of prediction	Shift	Dataset (1=simulated, 2=noisy simulated, 3=real)	Additionnal notes	Prediction = input + output (delta)	Average RMSE	Average error at rebound point	Average error at hit point	RMSE after 0,01s	RMSE after 0,05s	RMSE after 0,1s	RMSE after 0,2s	RMSE after 0,29s
LSTM-TAE	Seq2seq	L16D400	0.0002	128	500	1800	4000	130		2			0.075	0.090	0.078	0.065	0.072	0.080	0.098	0.114
LSTM-TAE	Seq2seq	L32D400	0.0003	128	500	1800	4000	130		2			0.108	0.121	0.107	0.108	0.112	0.114	0.122	0.142
GRU-TAE	Seq2seq	G16G3	0.0003	128	500	1800	4000	130		2			0.055	0.055	0.055	0.029	0.034	0.038	0.067	0.085
RNN	Recursive	R8	0.0020	128	1000	1800	10000	10	1	1		O	0.076	0.064	0.080	0.015	0.028	0.045	0.083	0.106
RNN	Recursive	R32	0.0010	128	500	1800	10000	10	1	1		O	0.107	0.077	0.079	0.009	0.026	0.042	0.109	0.143
RNN	Recursive	R128	0.0010	128	500	1800	10000	10	1	1		O	0.081	0.067	0.090	0.006	0.022	0.043	0.080	0.129
RNN	Recursive	R32R32	0.0010	128	500	1800	10000	10	1	1		O	0.103	0.062	0.083	0.006	0.025	0.042	0.077	0.119
RNN	Recursive	R32R32	0.0010	128	500	1800	10000	10	1	2		O	0.123	0.174	0.147	0.007	0.025	0.066	0.242	0.406
RNN	Recursive	R8	0.0010	128	500	1800	10000	10	1	2		O	0.078	0.072	0.086	0.020	0.044	0.055	0.088	0.118
RNN	Recursive	R32	0.0010	128	500	1800	10000	10	1	2		O	0.071	0.063	0.080	0.008	0.026	0.045	0.078	0.120
RNN	Recursive	R128	0.0010	128	500	1800	10000	10	1	2		O	0.100	0.057	0.075	0.008	0.022	0.040	0.075	0.093
RNN	Recursive	R8	0.0050	32	500	1800	500	10	1	3		O	0.116	0.114	0.118	0.033	0.048	0.073	0.137	0.214
RNN	Recursive	R32	0.0050	32	500	1800	500	10	1	3		O	0.113	0.073	0.081	0.022	0.024	0.046	0.093	0.143
RNN	Recursive	R128	0.0050	32	500	1800	500	10	1	3		O	0.066	0.104	0.097	0.041	0.066	0.083	0.124	0.159
RNN	Recursive	R32R32	0.0050	32	500	1800	500	10	1	3		O	0.102	0.079	0.081	0.017	0.018	0.042	0.100	0.160
Conv	Recursive	C64	0.0005	128	500	1800	10000	10	1	2		O	0.186	0.188	0.185	0.007	0.029	0.081	0.252	0.440
Conv	Recursive	C512	0.0005	128	500	1800	10000	10	1	2		O	0.216	0.224	0.239	0.007	0.031	0.091	0.297	0.546
Conv	Recursive	C16	0.0005	128	500	1800	10000	10	1	2		O	0.222	0.165	0.202	0.007	0.025	0.069	0.220	0.396

4.8 References

[1] R. L. Anderson, *A robot ping-pong player: experiment in real-time intelligent control*. MIT press, 1988. Accessed: Jun. 19, 2024. [Online]. Available: https://dl.acm.org/doi/abs/10.5555/42334

[2] Y. Huang, B. Schölkopf, and J. Peters, "Learning optimal striking points for a ping-pong playing robot," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 4587–4592. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7354030

[3] J. Tebbe, Y. Gao, M. Sastre-Rienietz, and A. Zell, "A Table Tennis Robot System Using an Industrial KUKA Robot Arm," in *Pattern Recognition*, vol. 11269, T. Brox, A. Bruhn, and M. Fritz, Eds., in Lecture Notes in Computer Science, vol. 11269. , Cham: Springer International Publishing, 2019, pp. 33–45. doi: 10.1007/978-3-030-12939-2_3.

[4] D. B. D'Ambrosio *et al.*, "Achieving Human Level Competitive Robot Table Tennis," Aug. 09, 2024, *arXiv*. Accessed: Aug. 26, 2024. [Online]. Available: http://arxiv.org/abs/2408.03906

[5] L. Acosta, J. J. Rodrigo, J. A. Mendez, G. N. Marichal, and M. Sigut, "Ping-pong player prototype," *IEEE robotics & automation magazine*, vol. 10, no. 4, pp. 44–52, 2003.

[6] Y. Zhang, W. Wei, D. Yu, and C. Zhong, "A tracking and predicting scheme for ping pong robot," *Journal of Zhejiang University SCIENCE C*, vol. 12, no. 2, pp. 110–115, 2011.

[7] A. Kyohei, N. Masamune, and Y. Satoshi, "The ping pong robot to return a ball precisely," *Omron Technics*, vol. 51, pp. 1–6, 2020.

[8] Y. Zhao, R. Xiong, and Y. Zhang, "Rebound modeling of spinning ping-pong ball based on multiple visual measurements," *IEEE Transactions on Instrumentation and Measurement*, vol. 65, no. 8, pp. 1836–1846, 2016.

[9] C. H. Lai and T. I. J. Tsay, "Self-Learning for a Humanoid Robotic Ping-Pong Player," *Advanced Robotics*, vol. 25, no. 9–10, pp. 1183–1208, Jan. 2011, doi: 10.1163/016918611X574678.

[10] J. Tebbe, L. Klamt, Y. Gao, and A. Zell, "Spin detection in robotic table tennis," in *IEEE international conference on robotics and automation (ICRA)*, 2020, pp. 9694–9700. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9196536/

[11] T. Gossard, J. Tebbe, A. Ziegler, and A. Zell, "SpinDOE: A ball spin estimation method for table tennis robot," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 5744–5750. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10342178

[12] S. Gomez-Gonzalez, S. Prokudin, B. Schölkopf, and J. Peters, "Real time trajectory prediction using deep conditional generative models," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 970–976, 2020.

[13] K. Mülling, J. Kober, and J. Peters, "A biomimetic approach to robot table tennis," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 1921–

1926. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5650305/

[14] Y. Gao, J. Tebbe, and A. Zell, "A model-free approach to stroke learning for robotic table tennis," in *International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2022, pp. 1–8. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9892776/

[15] Y. Zhao, R. Xiong, and Y. Zhang, "Model based motion state estimation and trajectory prediction of spinning ball for ping-pong robots using expectation-maximization algorithm," *Journal of Intelligent & Robotic Systems*, vol. 87, no. 3, pp. 407–423, 2017.

[16] K. M. Kulkarni and S. Shenoy, "Table tennis stroke recognition using two-dimensional human pose estimation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 4576–4584. Accessed: Jun. 20, 2024. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2021W/CVSports/html/Kulkarni_Table_Tennis_Stroke_Recognition_Using_Two-Dimensional_Human_Pose_Estimation_CVPRW_2021_paper.html?ref=https://githubhelp.com

[17] P. Blank, J. Hoßbach, D. Schuldhaus, and B. M. Eskofier, "Sensor-based stroke detection and stroke type classification in table tennis," in *ACM International Symposium on Wearable Computers - ISWC '15*, 2015, pp. 93–100. doi: 10.1145/2802083.2802087.

[18] R. L. Andersson, "Aggressive trajectory generator for a robot ping-pong player," *IEEE Control Systems Magazine*, vol. 9, no. 2, pp. 15–21, 1989.

[19] M. Matsushima, T. Hashimoto, and F. Miyazaki, "Learning to the robot table tennis task-ball control & rally with a human," in *2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme-System Security and Assurance (Cat. No. 03CH37483)*, pp. 2962–2969. Accessed: Aug. 21, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1244342/

[20] M. Matsushima, T. Hashimoto, M. Takeuchi, and F. Miyazaki, "A learning approach to robotic table tennis," *IEEE Transactions on robotics*, vol. 21, no. 4, pp. 767–771, 2005.

[21] H.-I. Lin and Y.-C. Huang, "Ball trajectory tracking and prediction for a ping-pong robot," in *9th International Conference on Information Science and Technology (ICIST)*, IEEE, 2019, pp. 222–227. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8836894/

[22] H. Li, H. Wu, L. Lou, K. Kühnlenz, and O. Ravn, "Ping-pong robotics with high-speed vision system," in *12th International Conference on Control Automation Robotics & Vision (ICARCV)*, IEEE, 2012, pp. 106–111. Accessed: Sep. 23, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6485142/

[23] E. Wu and H. Koike, "FuturePong: Real-time Table Tennis Trajectory Forecasting using Pose Prediction Network," in *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, Honolulu HI USA: ACM, Apr. 2020, pp. 1–8. doi: 10.1145/3334480.3382853.

[24] A. Van Den Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," in *International conference on machine learning*, PMLR, 2016, pp. 1747–1756. Accessed: Jun. 20, 2024. [Online]. Available: https://proceedings.mlr.press/v48/oord16.html

[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[26] A. Borovykh, S. Bohte, and C. W. Oosterlee, "Conditional Time Series Forecasting with Convolutional Neural Networks," Sep. 17, 2018, *arXiv*. Accessed: Jun. 20, 2024. [Online]. Available: http://arxiv.org/abs/1703.04691

[27] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, 1995, Accessed: Jul. 09, 2024. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=e26cc4a1c717653f323715d751c8dea7461aa105

[28] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," Dec. 11, 2014, *arXiv*. Accessed: Jul. 09, 2024. [Online]. Available: http://arxiv.org/abs/1412.3555

[29] A. Harsha, "The Ultimate Showdown: RNN vs LSTM vs GRU – Which is the Best? - Shiksha Online." Accessed: Jun. 20, 2024. [Online]. Available: https://www.shiksha.com/online-courses/articles/rnn-vs-gru-vs-lstm/

[30] R. Yu, S. Zheng, A. Anandkumar, and Y. Yue, "Long-term Forecasting using Higher Order Tensor RNNs," Aug. 23, 2019, *arXiv*. Accessed: Jun. 20, 2024. [Online]. Available: http://arxiv.org/abs/1711.00073

[31] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014, Accessed: Jul. 09, 2024. [Online]. Available: https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html

[32] P. Liu, X. Sun, Y. Han, Z. He, W. Zhang, and C. Wu, "Arrhythmia classification of LSTM autoencoder based on time series anomaly detection," *Biomedical Signal Processing and Control*, vol. 71, p. 103228, 2022.

[33] Y. Wei, J. Jang-Jaccard, W. Xu, F. Sabrina, S. Camtepe, and M. Boulic, "LSTM-autoencoder-based anomaly detection for indoor air quality time-series data," *IEEE Sensors Journal*, vol. 23, no. 4, pp. 3787–3800, 2023.

[34] X. Liu *et al.*, "Time-series forecasting based on fuzzy cognitive maps and GRU-autoencoder," *Soft Computing*, pp. 1–17, 2023.

[35] K. Merkelbach, S. Schaper, C. Diedrich, S. J. Fritsch, and A. Schuppert, "Novel architecture for gated recurrent unit autoencoder trained on time series from electronic health records enables detection of ICU patient subgroups," *Scientific reports*, vol. 13, no. 1, p. 4053, 2023.

[36] M. Abboush, C. Knieke, and A. Rausch, "GRU-based denoising autoencoder for detection and clustering of unknown single and concurrent faults during system integration testing of automotive software systems," *Sensors*, vol. 23, no. 14, p. 6606, 2023.

[37] D. Chen, H. Wang, and M. Zhong, “A short-term traffic flow prediction model based on AutoEncoder and GRU,” in *12th International Conference on Advanced Computational Intelligence (ICACI)*, IEEE, 2020, pp. 550–557. Accessed: Sep. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9177506/

[38] K. Cho, “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014.

[39] N. Qadeer, J. H. Shah, M. Sharif, M. A. Khan, G. Muhammad, and Y.-D. Zhang, “Intelligent tracking of mechanically thrown objects by industrial catching robot for automated in-plant logistics 4.0,” *Sensors*, vol. 22, no. 6, p. 2113, 2022.

[40] A. Vaswani, “Attention is all you need,” *Advances in Neural Information Processing Systems*, 2017, Accessed: Sep. 20, 2024. [Online]. Available: https://user.phil.hhu.de/~cwurm/wp-content/uploads/2020/01/7181-attention-is-all-you-need.pdf

[41] N. Wu, B. Green, X. Ben, and S. O’Banion, “Deep transformer models for time series forecasting: The influenza prevalence case,” *arXiv preprint*, 2020, Accessed: Sep. 20, 2024. [Online]. Available: https://arxiv.org/abs/2001.08317

[42] M. Abadi *et al.*, “TensorFlow: Large-scale machine learning on heterogeneous systems.” Mountain View, CA: Tensorflow, 2015.

[43] F. Chollet, “Building autoencoders in keras,” *The Keras Blog*, vol. 14, 2016, Accessed: Oct. 29, 2024. [Online]. Available: https://www.academia.edu/download/56975530/Building_Autoencoders_in_Keras.pdf

[44] S. K. Lam, A. Pitrou, and S. Seibert, “Numba: a LLVM-based Python JIT compiler,” in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, Austin Texas: ACM, Nov. 2015, pp. 1–6. doi: 10.1145/2833157.2833162.

[45] B. Toussaint, *GitHub - LiBRTy-Lab/GRU-TAE_for_trajectory_prediction:* (2024). Accessed: Oct. 29, 2024. [Online]. Available: https://github.com/LiBRTy-Lab/GRU-TAE_for_trajectory_prediction

[46] A. Nakashima, Y. Kobayashi, Y. Ogawa, and Y. Hayakawa, “Modeling of rebound phenomenon between ball and racket rubber with spinning effect,” in *ICCAS-SICE*, IEEE, 2009, pp. 2295–2300. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5335158/

[47] T. Rémond, “Physique du rebond: application à la balle de tennis de table,” PhD Thesis, Ecole normale supérieure de lyon-ENS LYON, 2023. Accessed: Jun. 20, 2024. [Online]. Available: https://theses.hal.science/tel-04221365/

[48] W. McKinney, “pandas: a foundational Python library for data analysis and statistics,” *Python for high performance and scientific computing*, vol. 14, no. 9, pp. 1–9, 2011.

CHAPITRE 5 ARTICLE 2 : BALL TRAJECTORY PREDICTION IN TABLE TENNIS BY COMBINING MACHINE LEARNING AND PHYSICAL MODELING

Submitted to *Applied Intelligence*, on March 28, 2025.

Abstract: Table tennis with collaborative robots has been a challenge in robotics for decades, due to its unique challenges, primarily requiring precise real-time ball trajectory predictions to enable responsive, accurate gameplay. Traditional physical models have been widely studied, but struggle with factors like spin, limiting accuracy.

With advancements in computational power, data-driven architectures such as gated-recurrent-unit encoder-decoder (GRU-ED), GRU trajectory encoder (GRU-TAE), and trajectory encoder (TAE) have shown improved accuracies in other real-time applications. However, for ball trajectory prediction in table tennis, there still has not been:

1. any comprehensive comparison between top-performing, data-driven and physical model-based methods.

2. any combination between machine learning and physical modeling, despite its potential benefits.

This paper presents a method for ball trajectory prediction in table tennis, combining machine learning and a physical model. Differentiable models were developed for parameter identification and validated on real-world and simulated trajectories. This approach was then compared with leading data-driven and traditional model-based methods.

The optimized model-based method with spin estimation (MBS^{+}) reduced errors by 42% over literature-based parameters and outperformed GRU-TAE by 9%. These results enhance robotic arm performance in returning balls to human players, allowing earlier positioning with fewer adjustments, moving toward real-world deployment in collaborative table tennis robots.

Additionally, combining the optimized model in an hybrid approach, averaging GRU-TAE, TAE, and MBS^{+}, achieved an additional 16% reduction in prediction errors and show the interest of combining data-driven with model-based methods. Future works will further optimize hybrid prediction methods and applying them to real-world ping-pong robots.

Keywords: robotics, machine learning, neural networks, trajectory prediction, table tennis, parameter optimization

5.1 Introduction

Trajectory prediction is a critical aspect of robotics, especially in dynamic, high-speed applications where real-time predictions of future physical quantities are required based on past measurements, such as in sports analysis.

Table tennis has been a prominent topic in robotics research for decades due to its unique combination of challenges, including real-time sensing, intelligent decision-making, motion control, and robotic design. Since the development of the first ping-pong playing robot in 1988 [1], several robotic systems have been developed, employing industrial serial robotic arms [2], [3], Cartesian systems [4], [5], parallel robots [6] or even humanoid systems [7], [8]. A crucial aspect for these robots is accurately predicting the future trajectory of a ball based on previous visual observations [9]. These visual measurements are often noisy, containing outliers and sparse data [10]. Additionally, the computation time – or latency – of the predictions plays a major role, as it must be significantly faster than the ball actual motion [11]. If trajectory prediction in table tennis is a research area, predicting trajectories can also be applied to other sports, such as tennis [12], or soccer [13]. Beyond sports, dynamic high-speed robotics frequently require the real-time prediction of future physical states, based on previous measurements.

Methods for predicting the trajectory of a ping-pong ball have been widely studied and can be categorized into two primary approaches:

1. Model-based approaches: grounded in well-established physical models known for their accuracy. Model-based methods typically involve two main steps. First, the position, velocity, and sometimes the spin of the ball are estimated based on observations. Then, the trajectory is predicted by solving the system of differential equations derived from the dynamic model. [14]
2. Data-driven approaches: the ball trajectory is directly inferred from previous observations without considering the dynamic model.

Figure 1 provides an overview of these methods, further detailed in the following sections.

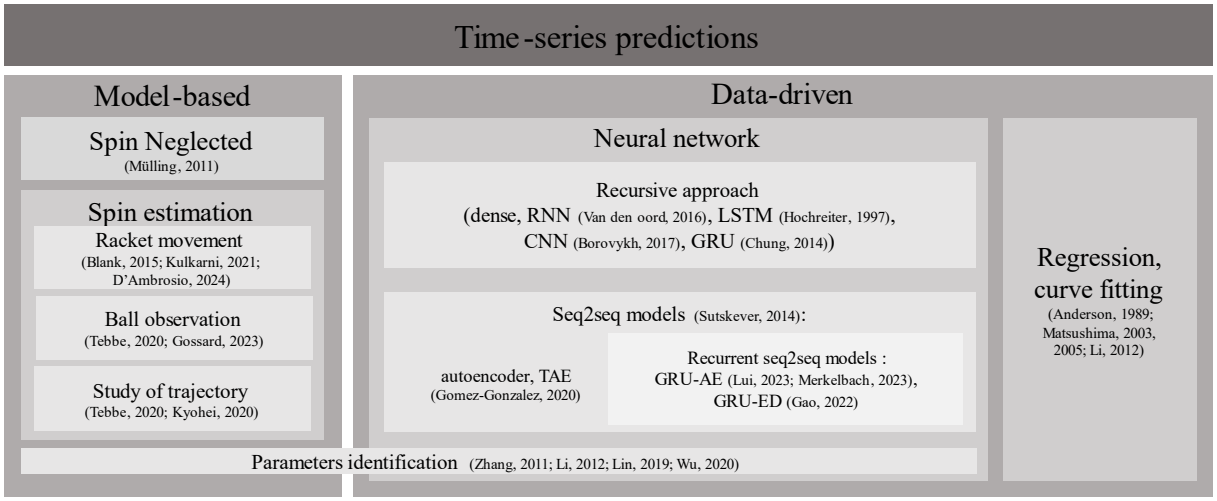


Figure 5.1 : Structure of existing methods for ping-pong ball trajectory prediction

5.1.1 Model-based methods

Model-based methods are often preferred for modeling and predicting trajectories in high-speed robotic systems [15], [16]. These methods, although based on differential equations, are relatively fast to compute and are grounded in well-established physical models known for their accuracy. However, a key limitation arises when certain essential parameters remain difficult to estimate, despite extensive study of the physical models [7].

In table tennis, spin is a crucial factor, necessary for human players to win competitions. This is because a spin in the ball results in a large change in the trajectory, especially when the ball bounces on the table [9]. However, accurately calculating ball spin in real-time from standard camera images is not feasible [6].

Because measuring the ball spin is challenging, **some approaches omit spin from the trajectory prediction** [16]. However, neglecting the ball spin generally leads to a significant drop in performances. To mitigate this, ping-pong robots that disregard the ball spin often attempt to hit the ball immediately after it bounces [6] or instruct users to minimize spin during play.

Several methods have been explored to consider the ball rotation speed into dynamic models, and these can be categorized into three main approaches: **deriving spin from the racket movement** during the stroke, **observing the ball directly** [10] or **analyzing the ball trajectory** [6].

In [17], spin estimation is achieved through stroke classification, which **analyzes the player body and racket movements**. The OMRON Forpheus ping-pong robot employs visible markers on the racket for stroke classification, as shown in its demonstration video [18]. Classification can also be performed using inertial measurement unit (IMU) sensors [19], [20], but the limitation in these approaches is that they provide only a stroke classification, without directly quantifying the spin value.

Thus, **some studies have computed spin from direct ball observations** using ultra-fast cameras [9], [10]. By tracking either the ball's logo [21] or a specially designed pattern on the ball [10], rotation data can be obtained. However, this requires expensive equipment. Interestingly, [9] notes that trajectory fitting methods yield greater accuracy than logo tracking. In [10], spin evaluation is achieved within 0.06s using data from 10 camera frames. From their study, a dataset of 232 real ball trajectories has been built, containing true values of spin and position. However, not every trajectory includes observations after the rebound, so the dataset cannot be used easily to evaluate the degradation of prediction accuracy after the rebound.

Finally, **spin can also be estimated by analyzing the ball's trajectory**. Some studies have estimated based on the observed effects of the Magnus force, which is the force exerted on the ball due to its rotation [9]. To achieve this, the initial linear speed of the ball must be known. Since even small errors in the initial speed can significantly impact the Magnus force estimation, outlier filtering is often applied to improve accuracy [9]. An alternative method for estimating spin using dynamic models is presented in [6]. Instead of computing the Magnus force, this approach derives spin by fitting the curvature of the ball's trajectory. The main limitation of these methods is that they require more computation to compute the spin.

Beyond spin, precise parameter identification is crucial for accurate predictions in model-based methods. Some parameters are standardized by the rules of the International Table Tennis Federation (ITTF). For instance, the ball must be spherical, with a diameter of 40mm and a mass of 2.7g. Additional constraints include rigidity, sphericity, and rebound height, submitted to a specific variance within the same batch of balls [22]. Notably, there are tolerances for parameters such as the mass (ranging between 2.67g and 2.77g) and the coefficient of restitution (between 0.89 and 0.92 for a 0.3m free fall on a standard steel block). This restitution coefficient also decreases as impact speed increases [23], [24]. Additionally, each brand produces balls with unique

properties. Furthermore, when using infrared cameras to detect the ball, a reflective tape must be added to the ball, which alters most of the ball's dynamic properties.

5.1.2 Data-driven methods

Data-driven methods can consider the ball spin from its impact on the ball trajectory. In [3], [4], the dynamic model and the force analysis are not considered, and the future trajectory prediction is made by mapping from the previous observations using a polynomial fitting or a local linear regression. However, as the dynamic model of the ball is highly non-linear, these methods come with high systematic errors [11].

5.1.2.1 Neural network-based methods

Research on data-driven and machine learning methods has seen significant growth and has been widely applied to predict ping-pong ball trajectories, using various inputs and outputs. The three main approaches are **parameter identification**, **recursive architectures**, and **sequence-to-sequence models**, as described below.

Parameters identification focuses on learning specific parameters of the trajectory. In [25], two neural networks were used to determine the polynomial coefficients of the ball's trajectory before and after the bounce. In [5], a neural network was trained to directly predict the hit point between the ball and the racket using only four observations of the ball. However, the accuracy of these methods is limited since certain parameters remain unknown.

Recursive approaches predict successive ball positions directly throughout its trajectory by using a neural network iteratively to compute the next position based on previous observations. Specialized recurrent neural architectures such as recurrent neural networks (RNNs) [26], long-short term memory (LSTM) neural networks [27] or autoregressive models [28] have been employed for ping-pong ball trajectory prediction. Other recurrent methods, including convolutional neural networks (CNNs) [29] and gated-recurrent unit (GRU) [30] neural networks are also widely used for time series prediction. The key difference between RNNs, LSTMs, and GRUs lies in how they handle memory and dependencies [31]. However, recursive approaches suffer from cumulative errors, reducing accuracy over time [11], [27], [28], [32], [33].

Sequence to sequence (seq2seq) **models** address the issue of cumulative errors by avoiding the use of predictions as inputs for future steps [34]. In [11], a trajectory autoencoder (TAE) was used

for table tennis ball trajectory prediction and compared to an LSTM. The results showed better accuracy, particularly for long-term predictions, with an error three times lower for the TAE compared to the LSTM. However, the main limitation of TAE is that their architecture is not specialized for time-series prediction, because it does not include specific recursive layers. As a result, in [33] and [14], recurrent seq2seq models, combining recursive layers and seq2seq models, were applied to trajectory prediction, specifically a GRU-TAE and a GRU encoder-decoder (GRU-ED). [33] demonstrated a 14% improvement in prediction accuracy compared to traditional seq2seq models using a dataset of real ball trajectories. However, this study also showed that all data-driven methods still face challenges, as for accurately predicting the ball behavior near the rebound.

5.1.3 Problem and objective

Following Sections 1.1 and 1.2, several studies [11], [14], [33] showed that combining machine learning and model-based approaches could improve the accuracy of the trajectory prediction. However, to the best of our knowledge, no method currently integrates both machine learning and model-based techniques within a single prediction algorithm.

Additionally, while both model-based and data-driven methods have been widely studied for ping-pong ball trajectory prediction, most research has focused on either data-driven methods, physical models or parameter estimation. When comparisons are made between data-driven and physical-based methods, they are often done using a classical, non-optimized model [11]. As a result, there is no standardized comparison between the latest data-driven techniques and the most advanced physical models.

The objective of this study is to develop a method for ball trajectory predictions in table tennis, combining both machine learning methods and physical models. A subsequent objective is to compare the accuracy between the best current methods existing in the literature, including both data-driven and model-based approaches.

To achieve these objectives, a model-based trajectory prediction algorithm using differentiable models to optimize the ball's dynamic parameters is developed. This prediction method is then compared with the more accurate data-driven methods identified in [33] as well as traditional physical model-based methods.

Additionally, this method is combined with a GRU-TAE and a TAE to show the interest of

combining whole data-driven with model-based methods that could be explored in future works.

5.2 Methodology

The method section is organized as follows: first, architectures compared with our method are detailed (section 2.1). Secondly, the process used for the parameter identification is described (section 2.2). Thirdly, the process used for the construction of the datasets are detailed (section 2.3). Finally, the evaluation strategy is explained (section 2.4).

5.2.1 Reference architectures

From the state-of-the-art, notable methods have been identified as references for comparison with our optimized model. These methods fall into two primary categories: model-based and data-driven approaches.

5.2.1.1 Data-driven methods

The key data-driven methods have been compared in [33]. GRU-TAE and TAE have been identified as the most accurate data-driven prediction models. Their implementations are detailed in [33], a summary of these implementations is available in the appendix. The results presented in this paper were generated using the same datasets as in that study.

5.2.1.2 Model-based methods

To evaluate the accuracy of our method, traditional model-based architectures were implemented. These are all based on well-studied equations of the dynamic model. They are presented in equations **(1) (2)** and **(3).** Equation **(1)** describes the flying phase of the ball.

$$m\ddot{\boldsymbol{x}}(\boldsymbol{t}) = -m\boldsymbol{g} - m\ddot{\boldsymbol{x}}_{\boldsymbol{D}}(t) + m\ddot{\boldsymbol{x}}_{\boldsymbol{M}}(t) = -m\boldsymbol{g} - C_D\|\dot{\boldsymbol{x}}(t)\|\dot{\boldsymbol{x}}(t) + C_M\{\dot{\boldsymbol{x}}(t) \times \boldsymbol{\omega}(\boldsymbol{t})\} \quad (\mathbf{1})$$

Where $x(t), \dot{x}(t)$ and $\ddot{x}(t)$ are respectively the ball position [m], velocity [m/s] and acceleration [m/s²], at time t in the global reference frame. ω represents the spin of the ball [rad/s], m the mass of the ball [g], $m\ddot{x}_D$ and $m\ddot{x}_M$ the drag and Magnus forces, C_D [g/m] and C_M [m²/s] the drag and Magnus constants.

Equations **(2)** and **(3)** are used to compute the rebound phenomenon, from [22]. Note that these equations are valid from a rolling ball without sliding, i.e., with a small speed. Value used for the parameters have been obtained experimentally in [22] using a ping-pong ball in ABS plastic –

parameters are different with a celluloid ball. Equations describing the sliding phase can also be found in [22].

$$\dot{\boldsymbol{x}}' = \boldsymbol{A_v}\dot{\boldsymbol{x}} + \boldsymbol{B_v}\boldsymbol{\omega} \quad (\mathbf{2})$$

$$\boldsymbol{\omega}' = \boldsymbol{A_w}\dot{\boldsymbol{x}} + \boldsymbol{B_w}\boldsymbol{\omega} \quad (\mathbf{3})$$

With

$$\boldsymbol{A_v} = \begin{bmatrix} \frac{3}{5} & 0 & 0 \\ 0 & \frac{3}{5} & 0 \\ 0 & 0 & -C_{RZ} \end{bmatrix}, \boldsymbol{B_v} = \begin{bmatrix} 0 & -\frac{2r}{5} & 0 \\ \frac{2r}{5} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \boldsymbol{A_w} = \begin{bmatrix} 0 & \frac{3}{5r} & 0 \\ \frac{-3}{5r} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \boldsymbol{B_w} = \begin{bmatrix} 2/5 & 0 & 0 \\ 0 & 2/5 & 0 \\ 0 & 0 & C_{\omega z} \end{bmatrix}$$

$$C_{\omega z} = 1 - \frac{3\tau cb}{2r^2}\frac{\|\vec{v}^P\|}{\|\vec{v}^M\|}|\dot{x}_z|^{0.75}u$$

With $u = \mu(1 + C_{rz})\frac{|\dot{x}_z|}{\|\vec{v}^P\|}$, $\|\vec{v}^M\| = \sqrt{(\omega_z\tau)^2 + a^2}$. With $\mu = 0.22$ the dimensionless dynamic coefficient of friction between the ball and table, $C_{RZ} = 0.9$ the dimensionless coefficient of restitution of the table in the z direction, $\tau = 0.5ms$ the duration of the ball/table contact, $r = 0.02m$ the radius of the ball and $\vec{\boldsymbol{v}}^{\boldsymbol{P}}$ the tangential velocity ball/table defined by $\vec{\boldsymbol{v}}^{\boldsymbol{P}} = [\dot{x}_x - r\omega_y\ ;\ \dot{x}_y + r\omega_x\ ;\ 0]$.

The first baseline "model-based" method (referred to as "MB") predicts the ball trajectory without considering spin. It has been implemented using simplified equations (1) to (3). The initial position is taken from the first detected point of the trajectory, and the initial speed is computed iteratively using the first and last measured points before the rebound. The spin is neglected and set to 0.

The second method, "model-based with spin estimation" (referred to as "MBS") is inspired by [6]. It is like the MB method but incorporates ball spin. Once the initial position and linear velocity are computed, the spin is estimated from its impact on the trajectory, by minimizing the following loss function:

$$L(\omega) = \sum_{i=1}^{N} e_i(\omega) \quad (\mathbf{4})$$

Where e_i is the error between the predicted and measured positions [m] at time i [s] and ω the spin of the ball [rad/s]. To reduce computation time, the optimal spin is computed by testing mapped values in parallel instead of finding it iteratively.

5.2.2 Parameter identification of the differentiable model

The parameters identification was conducted using 50 trajectories from either the simulated or the real dataset. This number of trajectories has been chosen experimentally to get robust results while limiting the quantity of data required for training.

The first step involved identifying parameters related to free flight. To achieve this, a differentiable model was built with Tensorflow [35], based on the model of equation (1). The trajectory of the ball can be defined only by two key coefficients: drag (C_D) and Magnus (C_M). The optimization of these two coefficients allows us to simplify the problem, as it also performs implicit approximations of other parameters, such as mass, which are included in this parameter value. For simplicity, we assumed $C_D = C_M$ and optimized only C_D while neglecting spin, i.e.$\omega = 0$. To train our model, the following loss function was applied to all trajectory data before the rebound.

$$L(C_D, \dot{x}) = \sum_{i=1}^{N} |e_i(C_D, \dot{x})| + \sum_{i=1}^{N} e_i(C_D, \dot{x})^2 \qquad (\mathbf{5})$$

Where e_i represents the difference between the predicted and the actual position of the i^{th} point – out of N – of the trajectory [m] and $\dot{x}(t)$ is the ball velocity [m/s]. Using, the GradientTape function and the Adam optimizer of Tensorflow [35], gradient descent was performed on both C_D (and thus C_M) and the initial velocities for all trajectories. The initial estimations for initial ball speed and position were calculated using a polynomial approximation. The optimization was run five times across all available trajectories, with the initial speed of the i^{th} trajectory updated after each computation, as well as the global C_D parameter.

The second step was to optimize parameters related to the rebound. To simplify this process, we focused on optimizing the parameters that affect the speed of the ball, specially A_v and B_v, linking velocities before and after the rebound, and expressed as follows:

$$\boldsymbol{A_v} = \begin{bmatrix} C_{RX} & 0 & 0 \\ 0 & C_{RX} & 0 \\ 0 & 0 & -C_{RZ} \end{bmatrix}, \boldsymbol{B_v} = \begin{bmatrix} 0 & -C_{RW}r & 0 \\ C_{RW}r & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

In this way, the parameters C_{RX}, C_{RZ} and C_{RW} are sufficient to determine the speed of the ball after the rebound. Once again, this formulation allows us to simplify the problem, as it performs implicit approximations of other rebound related parameters. Using the previously optimized C_D and C_M, the initial speeds for all trajectories were calculated with the method described in the previous section, using all observations before the rebound. Once again, spin was neglected. Knowing the initial parameters for each trajectory, a second differentiable model was trained to optimize the rebound coefficients, C_{RX} and C_{RZ}. The loss function used to train this model was:

$$L(C_D, \dot{x}) = e_{last}(C_{RX}, C_{RZ})^2 \quad (\mathbf{6})$$

Where e_{last} represents the difference between the predicted and actual position at the final point of the trajectory [m]. Using, GradientTape function and Adam optimizer from Tensorflow [35], gradient descent was applied on both C_{RX} and C_{RZ}, utilizing previously computed velocities. The optimization was run five times across all available trajectories, with C_{RX} and C_{RZ} optimized using two separate optimizers.

Finally, C_{RW} was optimized. The initial speed and ball spin for all trajectories were recalculated using the previously optimized parameters and all available observations before the rebound. Knowing these initial parameters for each trajectory, a final differentiable model was trained to optimize C_{RW}. The loss function used was:

$$L(C_{RW}) = e_{last}(C_{RX}, C_{RZ})^2 + \sum_{i=1}^{N} e_i(C_{RX}, C_{RZ})^2 \quad (\mathbf{7})$$

Where e_{last} is the difference between the predicted and actual positions at the final point of the trajectory [m] and e_i the difference between the predicted and actual positions of the i^{th} point of the trajectory [m]. Once again, GradientTape function and Adam optimizer from Tensorflow [35], were used to perform gradient descent on C_{RW}, based on the previously determined initial parameters. The optimization was run five times across all available trajectories.

Here below, the abbreviations "MB^{+}" and "MBS^{+}" will be used when referring to the optimized parameters applied to the MB and MBS methods, respectively.

We provide access to the code used for this optimization in [36].

5.2.3 Trajectory dataset construction

To evaluate the accuracy of the compared methods, both simulated (with and without noise) and real ball trajectories were analyzed.

First, for simulated ball trajectories, the equations of the dynamic model presented in equations **(1)** to **(3)** were used to generate trajectories, employing the parameters from [22].

To assess prediction accuracy in the simulation, we generated 2000 random trajectories with randomly initialized parameters – position, translational velocity, and rotational velocity –. The trajectories were generated by using the model described in equations **(1)** to **(3)**, with each 1 second, at 200 frames per second (fps).

Of these 2000 simulated trajectories, 1800 were used to train the data-driven methods, while the remaining 200 were reserved for the test set. This number of trajectories is the same as [11] and allows robust results while preserving an acceptable amount of required data.

Secondly, for real ball trajectories, three Optitrack infrared cameras, capturing at 100 fps and a ping-pong ball covered with infrared tape were used. To gather enough data to train the models, over 300 real trajectories were captured. These trajectories included variable spin, i.e. topspin and backspin trajectories, to generate a good variability in the data, all with a bounce on the table. To ensure consistent evaluation conditions between model-based and data-driven methods, the real dataset was split into training and test sets. The test set consisted of 40 trajectories – number chosen experimentally to be sufficient to get small variance in the results – while the remaining data formed the training set.

We provide access to both the simulated and real datasets in [36].

5.2.4 Evaluation strategy

First, to evaluate the performance of the parameter identification and its impact, identification was first performed using a simulated dataset built with known parameters. Secondly, to analyze the effect of incorrect parameter values on trajectory prediction accuracy, our methods were tested using different sets of parameters, still in simulation. Thirdly, the effectiveness of parameter identification on the real dataset was measured by comparing the accuracy between optimized and non-optimized parameter sets, i.e MBS and MBS+ methods.

Accuracy was assessed by computing the RMSE across all predicted trajectories in the test set, for all numbers of observations. The RMSE was calculated as $e = \sqrt{\frac{1}{N}\sum(\boldsymbol{x} - \boldsymbol{x_p})^2}$ where x represents the actual ball trajectory and x_p the predicted trajectory.

The prediction accuracy was then compared with traditional model-based methods, using both optimized and non-optimized parameters, as well as the best data-driven methods from [33], i.e., the TAE and GRU-TAE. The implementation of TAE and GRU-TAE, and information on the choice and optimization of hyperparameters, are provided as supplementary material to this paper. These prediction methods are summarized in Table 1.

Finally, predictions of trajectories were computed, simply averaging the predictions of MBS+, GRU-TAE and TAE, to show the potential of further combining the best data-driven methods with optimized model-based methods, as follows:

$$\boldsymbol{x_p} = \left(\boldsymbol{x_{pMBSopti}} + \boldsymbol{x_{pGRUTAE}} + \boldsymbol{x_{pTAE}}\right)/\boldsymbol{3} \quad (\boldsymbol{8})$$

Accuracy was then assessed by computing the RMSE across all predicted trajectories in the test set, for all numbers of observations. The RMSE was calculated as $e = \sqrt{\frac{1}{N}\sum(\boldsymbol{x} - \boldsymbol{x_p})^2}$, where $\boldsymbol{x}$ represents the actual ball trajectory and $\boldsymbol{x_p}$ the predicted trajectory.

Finally, prediction accuracy was compared with traditional model-based methods, using both optimized and non-optimized parameters, as well as the best data-driven methods from [33], i.e., the TAE and GRU-TAE. These prediction methods are summarized in Table 1.

Tableau 5.1 : List of prediction methods used for comparison for ping-pong ball trajectory prediction

Family		Method	Abbreviation
From literature	Data-driven methods	Gated recurrent unit trajectory autoencoder	GRU-TAE
		Trajectory autoencoder	TAE
	Model-based methods	Model-based, no spin	MB
		Model-based, spin	MBS
Developed methods	**Combined methods**	**Model-based, no spin, with optimized parameters**	**MB⁺**
		Model-based, spin, with optimized parameters	**MBS⁺**

Additional methods tested for exploration	Hybrid methods	Average of GRU-TAE and MBS^{+}	Hybrid^{2}
		Average of GRU-TAE, TAE and MBS^{+}	Hybrid^{3}

5.3 Results

5.3.1 Parameters identification

Table 2 presents the results of parameter identifications under three conditions: simulated data without clean observations, simulated data with noisy observations, and real-world data.

Tableau 5.2 : Results of parameters optimizations

Parameter	True values of simulation	Identified parameters with clean observations (error in %)	Identified parameters with noisy observations (error in %)	Identified parameters with real-world observations
C_M / C_D	0.39	0.401 (3%)	0.414 (6%)	0.331
C_{RZ}	0.9	0.895 (<1%)	0.895 (<1%)	0.805
C_{RX}	0.6	0.663 (11%)	0.662 (10%)	0.602
C_{RW}	0.4	0.349 (13%)	0.358 (11%)	-0.012

First, using clean observation without noise in the simulated dataset, the identified drag and Magnus parameters were 0.401, corresponding to a 3% error compared to their true values. The rebound coefficients C_{RZ} and C_{RX} were estimated to 0.895 and 0.663, respectively, with errors of less than 1% and 11%, respectively. Lastly, the coefficient C_{RW} was estimated at 0.349, representing a 13% error.

Secondly, when using noisy observations, the identified drag and Magnus parameters were 0.414, corresponding to a 6% error. The rebound coefficients C_{RZ} and C_{RX} were estimated to 0.895 and 0.662, with errors less than 1% and 11%, respectively. The coefficient C_{RW} was estimated at 0.358, corresponding to a 11% error.

Thirdly, using the real-world dataset, the identified drag and Magnus force coefficients were 0.331. The optimized rebound coefficients C_{RZ} and C_{RX} were 0.805 and 0.602, respectively, while the C_{RW} coefficient was -0.012.

5.3.2 Impact of dynamic parameters

Table 3 presents the global RMSE for trajectory predictions using various sets of parameters, based on both clean and noisy observations.

Tableau 5.3 : Impact of the parameter accuracy on the trajectory prediction accuracy

Set of parameters	Global RMSE for clean observations (mm)	Global RMSE for noisy observations (mm)
Values from the literature [19] (equal to the set used to generate simulated data)	24	31
10% error drag and Magnus coefficients ($\boldsymbol{C_D}$ & $\boldsymbol{C_M}$)	36 (+50%)	42 (+35%)
25% error drag and Magnus coefficients ($\boldsymbol{C_D}$ & $\boldsymbol{C_M}$)	46 (+92%)	54 (+74%)
10% error rebound coefficients ($\boldsymbol{C_{RX}}$, $\boldsymbol{C_{RZ}}$)	38 (+58%)	45 (+45%)
25% error rebound coefficients ($\boldsymbol{C_{RX}}$, $\boldsymbol{C_{RZ}}$)	65 (+171%)	70 (+126%)
No spin impact on rebound ($\boldsymbol{C_{RW}}$ = 0)	67 (+179%)	68 (+119%)
10% error on all parameters ($\boldsymbol{C_D}$, $\boldsymbol{C_M}$, $\boldsymbol{C_{RX}}$, $\boldsymbol{C_{RZ}}$, $\boldsymbol{C_{RW}}$)	32 (+33%)	46 (+48%)
25% error on all parameters ($\boldsymbol{C_D}$, $\boldsymbol{C_M}$, $\boldsymbol{C_{RX}}$, $\boldsymbol{C_{RZ}}$) + $\boldsymbol{C_{RW}}$ = 0	88 (+266%)	88 (+184%)
Set of optimized parameters	31 (+29%)	35 (+13%)

The reference accuracy corresponds to the prediction method using the true value of dynamic parameters. The global RMSE for predictions is 24 mm for perfect data and 31 mm for noisy data. For a 10% and a 25% error in both the drag and Magnus coefficients, the global RMSE increases to 36 mm and 46 mm with clean data, representing a 50% and 92% degradation, respectively. With noisy data, the RMSE rises to 42 mm and 54 mm, showing a degradation of 35% and 74%, respectively, compared to predictions using the true parameter values. For a 10% and a 25% error in the rebound coefficients – C_{RX} and C_{RZ} – the global RMSE with clean data is 38 mm and 65 mm, representing a 58% and 171% degradation, respectively. With noisy observations, the RMSE is 45 mm and 70 mm, corresponding to a degradation of 45% and 126%, respectively. When C_{RW} is set to 0, i.e., neglecting the effect of spin on the speed during the rebound, the global RMSE is 67 mm for clean data and 68 mm for noisy data, representing a degradation of 179% and 119%, respectively. For a 10% and 25% error in all dynamic parameters – C_M, C_D, C_{RX} and C_{RZ} – the global RMSE with clean data is 32 mm and 88 mm, representing a degradation of 33% and 266% degradation, respectively. With noisy data, the RMSE is 46 mm and 88 mm, showing a 48% and 184% degradation, respectively. Finally, using the parameter values obtained from optimization – starting from random initial values – the RMSE is 31 mm for clean data and 35 mm for noisy data,

representing a degradation of 29% and 13%, respectively, compared to predictions with the true parameter values.

5.3.3 Validation of the parameter's identification

Table 4 illustrates the impact of parameter identification on the accuracy of real-world trajectory predictions.

Tableau 5.4 : Impact of the parameter optimization on real trajectory prediction accuracy

Set of parameters	Global RMSE for real-world observations (mm)	RMSE of the last predicted point of the trajectory (mm)
Values from the literature [19]	100	257
Values from the literature [19], $\boldsymbol{C_{RW} = 0}$	67 (-33%)	131 (-49%)
Drag and Magnus coefficients optimized ($\boldsymbol{C_D}$ & $\boldsymbol{C_M}$), $\boldsymbol{C_{RW} = 0}$	67 (-33%)	134 (-48%)
Rebound coefficients optimized ($\boldsymbol{C_{RX}}$, $\boldsymbol{C_{RZ}}$), $\boldsymbol{C_{RW} = 0}$	61 (-39%)	106 (-59%)
Optimized parameters ($\boldsymbol{C_D}$, $\boldsymbol{C_M}$, $\boldsymbol{C_{RX}}$, $\boldsymbol{C_{RZ}}$) + $\boldsymbol{C_{RW}}$ = 0	59 (-41%)	101 (-61%)
Set of optimized parameters	58 (-42%)	99 (-61%)

Our reference values are the overall RMSE and the average RMSE for prediction of the final point of the trajectory – using between 15 and 65 points for the prediction. Using the set of parameters from [22], the overall RMSE is 100 mm, and the RMSE for the last point is 257 mm. When the coefficient C_{RW} is set to 0, effectively neglecting the effect of spin on the speed during rebound, the RMSE values decrease to 67 mm and 131 mm, representing a 33% and 49% reduction in error, respectively. With optimized drag (C_D) and Magnus (C_M) coefficients, the overall RMSE is 67 mm, and the RMSE for the last point is 134 mm, representing improvements of 33% and 48%, respectively, compared to the values directly taken from the literature [22].

Optimizing only the rebound coefficients – C_{RX} and C_{RZ} – results in RMSE values of 61 mm and 106 mm, representing 39% and 59% improvements, respectively, over the reference. When all parameters are optimized except for C_{RW} = 0, the RMSE is 59 mm for the overall prediction and 101 mm for the final point, corresponding to improvements of 41 % and 61%, respectively.

Finally, using all optimized parameters, the RMSE values are 58 mm and 99 mm, indicating a reduction of 42% and 61%, respectively, compared to the initial parameter set.

5.3.4 Comparison of methods accuracy

Table 5 illustrates the accuracy of the prediction methods by comparing the average RMSE of all predicted test trajectories across three datasets: clean and noisy observations from simulated trajectories, and real trajectory data.

Tableau 5.5 Comparison of computation time for tested methods

Method used for prediction	**Real-world trajectories**	**Clean observations (simulation)**	**Noisy observation (simulation)**	**Average calculation time (ms)**
GRU-TAE	64	24	24	8.2 (around 5000s for training)
TAE	65 (+2%)	39 (+63%)	36 (+50%)	2 (around 200 s for training)
MB	72 (+13%)	66 (+175%)	65 (+171%)	1.8
MBS	100 (+56%)	24 (=)	31 (+29%)	43
MB^{+}	**65 (+2%)**			**1.8**
MBS^{+}	**58 (-9%)**	**31 (+29%)**	**35 (+46%)**	**43**
Hybrid[2]	52 (-19%)			51
Hybrid[3]	49 (-24%)			53

The reference GRU-TAE model achieves RMSE values of 64 mm, 24 mm and 24 mm, respectively, while the TAE got an accuracy of 65 mm, 39 mm and 36mm,

The reference method of the literature, MB, based on a dynamic model using the parameters from **[22]**, and neglecting spin, yields RMSE values of 72 mm for the real-world dataset, 66 mm with clean observations from simulated dataset, and 65mm with noisy observations from simulated dataset, representing degradations of 13%, 175% and 171%, respectively, compared to the GRU-TAE. Estimating the spin with the parameters from the literature (MBS method) results in an RMSE of 100 mm for the real-world dataset, indicating a 56% degradation compared to the GRU-TAE on the real-world dataset.

After optimizing parameters on real-world data using our method, the RMSE of the MB^{+} method decreases to 65 mm, representing a 10% improvement compared to MB method. In addition the

accuracy on real-world trajectory prediction in only 2% higher than the GRU-TAE. When a spin estimation is performed after our parameter optimization (MBS$^+$ method), the RMSE for the real-world dataset is reduced to 58 mm, representing a 9% improvement over the reference GRU-TAE.

Finally, the accuracy of the prediction using combination of both data-driven and model-based methods was also tested. Hybrid^2 method results in an RMSE of 52 mm, representing a 19% improvement over the GRU-TAE. Finally, Hybrid^3 method achieves an RMSE of 49 mm with the real-world dataset, representing a 24% improvement.

Table 5 also presents the computation time required for a single trajectory prediction. The model-based method that neglects spin (MB and MB$^+$) takes on average 1.8 ms to compute, while the model-based method that estimates the spin (MBS and MBS$^+$) requires on average 43 ms. For the data-driven methods, TAE and GRU-TAE, the computation times are 2 ms and 8.2 ms, respectively. Finally, when multiple prediction methods are combined, the total computation time is the sum of the method's time. For Hybrid^2 method, the computation time is in average 51 ms, and 53 ms for Hybrid^3.

5.4 Discussion

The first main result, shown in Table 4, demonstrates the **effectiveness of parameter optimization using differentiable models, leading to a 42% reduction in RMSE when using optimized parameters compared to those from literature**. Specifically, optimizing parameters related to the rebound of the ball is crucial for reducing RMSE. Even for parameters with which the optimization is difficult, i.e $\boldsymbol{C_{RW}}$, the optimization allows for reducing RMSE. For example, by using a value close to 0 resulting from the optimization for $\boldsymbol{C_{RW}}$ coefficient rather than using an incorrect value improves prediction accuracy by 33%. Optimizing all rebound-related coefficients leads to a 39% improvement. It is important to note that reference parameters from the literature **[22]**, do not consider the reflective tape on the ball, which significantly affects the ball's dynamics and explains the significant difference in prediction accuracy. Table 5 shows that the optimization is efficient on real-world trajectories without requiring changes in the proposed learning algorithm. The simulated results highlight the necessity of spin estimation for the model-based method. However, results with real-world trajectories in Table 5, show that spin estimation only improves accuracy when using optimized parameters.

The second main result illustrated by Table 5 is about the comparison of the accuracy of both model-based and data-driven prediction methods. Initial simulation results show that data-driven methods, especially GRU-TAE, achieve accuracy close to the one of MBS and MBS$^+$ methods. As in **[33]**, GRU-TAE slightly outperforms TAE for both simulated and real-world datasets. Table 5 demonstrates that **MBS$^+$ method provides better accuracy than GRU-TAE with a 9% improvement**. It is important to consider this improvement in light of the results from **[33]** which showed a better modeling of the trajectory using model-based method near the rebound, compared to TAE and GRU-TAE. This result is in contradiction with **[11]**, where TAE outperformed the model-based method. However, the shared code suggests that the Magnus effect was neglected, and rebound coefficients were not optimized.

The third main result is about the computation times of the tested methods, shown in Table 5. The conclusion is that data-driven methods, such as TAE and GRU-TAE can be used for real-time prediction requiring 2ms and 8.2ms per prediction, respectively. This computation time is quicker than the one of standard cameras usually used for real-world ping-pong robotic systems (around 100 Hz). They are faster than model-based methods including spin estimation, which require multiple trajectory computations while optimizing initial parameters. These take longer –43ms on average – and would need a powerful processor for real-time use. Additionally, the more observations used, the longer the calculation time. For real-world robotic systems, fewer observations are typically used for prediction, as in **[9]**, where 10 to 30 observations were sufficient for real-time use, which accelerate computation. Finally, when neglecting the spin, the model-based method takes 1.8ms on average, which allows an easy transfer into real-time applications. The degradation due to neglecting the spin (between MB+ and MBS+ methods) is only 12%. These results suggest that a trade between prediction accuracy and computation time can be done, whether including or neglecting the ball spin, depending on the available computational power, to ensure real-time trajectory prediction.

Additionally, the validity of the parameter identification process is presented in Table 2. Using the simulated dataset, the differentiable models accurately retrieve the value of parameters with an error margin of less than 13%, based solely on ball trajectories. In particular, the differentiable models perform well on parameters related to free flight – namely, the drag ($\boldsymbol{C_D}$) and Magnus effect ($\boldsymbol{C_M}$) coefficients – achieving errors below 6%, and for the rebound coefficient around the $\boldsymbol{z}$ axis – $\boldsymbol{C_{RZ}}$ – with an error under 1%. The model struggles more with identifying the other rebound

coefficients – $\boldsymbol{C_{RX}}$ and $\boldsymbol{C_{RW}}$ – showing errors between 10% and 13%. This is likely because these parameters depend heavily on the ball state during the rebound (speed and spin), and one must optimize the model parameters independently. Also, the parameter optimization using the simulated dataset demonstrates robustness against noisy observations. There is no significant difference in the accuracy of estimated parameters using clean or noisy observations. One advantage of differentiable models is that they can identify all desired parameters from real-game trajectories without requiring a specific setup for each parameter, unlike in **[37]** where 25 ball throws were needed to compute $\boldsymbol{C_{RZ}}$.

The importance of accurate parameter estimation is also demonstrated in Table 3. Using clean observations, the overall RMSE of trajectory predictions improves when correct parameter values are used – 24mm. A slight 10% error in all parameters increases the RMSE by 33%, and a 25% error results in a 266% increase compared to true parameters values.

Table 3 also shows that errors in $\boldsymbol{C_D}$ and $\boldsymbol{C_M}$ cause less degradation in prediction accuracy compared to errors in rebound coefficients $\boldsymbol{C_{RX}}$ and $\boldsymbol{C_{RZ}}$ – 50% versus 58% for a 10% error, and 92% versus 171% for a 25% error, using clean observations from the simulated dataset. This is because the optimization process compensates for $\boldsymbol{C_D}$ and $\boldsymbol{C_M}$ errors with initial parameters, such as the initial velocity while errors in $\boldsymbol{C_{RX}}$ and $\boldsymbol{C_{RZ}}$ only become apparent after the rebound. Similarly, neglecting the $\boldsymbol{C_{RW}}$ coefficient results in a 179% degradation in prediction accuracy. This explains why initial parameters are often computed using the first and last observations, as in **[6]**, to reduce the impact of noisy data and compensate for inaccuracies in parameters of the dynamic model.

The results from clean observations are reinforced by findings with noisy observations. The RMSE with perfect parameters is higher with noise – 31mm compared to 24mm without noise. However, the trend remains the same: the greater the error in parameter values, the higher the RMSE in trajectory predictions. Table 3 further shows that while using optimized parameters leads to a higher RMSE than perfect parameters—29% and 13% degradation for clean and noisy observations, respectively—the RMSE is still lower than for any 10% error in a single parameter, validating the method in simulation. Results on real-world trajectories, also based on observations with white noise, show that parameter optimization can be done successfully, and allows reducing the RMSE of trajectory prediction.

Finally, direct evaluation of the parameter optimization accuracy for the real ball is not feasible.

However, since our ball is covered with an infrared tape, it is possible to interpret identified parameter values. First, the drag and Magnus coefficients are smaller than those in the literature **[22]**, likely due to the increased mass of the ball from the infrared tape, which reduces drag. Similarly, the smaller $\boldsymbol{C_{RZ}}$ and $\boldsymbol{C_{RX}}$ coefficients can be attributed to the tape, which affects the restitution of speed during the rebound. Both $\boldsymbol{C_{RZ}}$ and $\boldsymbol{C_{RX}}$ are approximately 10% lower with the real ball compared to the simulation. Lastly, the $\boldsymbol{C_{RW}}$ coefficient on the real ball is close to zero, indicating the difficulty to accurately estimate spin and its impact on trajectories. This aligns with previous studies that only manage to classify shots, such as top-spin or backspin **[17], [19], [20]**. Other sources of uncertainty include table flatness and irregularities caused by the tape. Parameter estimation could likely be improved by using more trajectories during the model's optimization process, as this study only used 50 trajectories. The proposed method is not restricted to table tennis, as a standard ball was not used, and can be easily adapted to other sports or applications.

Finally, the results about hybrid methods combining data-driven and model-based methods allow to improve the prediction accuracy even by simply averaging predictions further improves accuracy. Hybrid² and Hybrid³ methods achieve a 10% and 16% reduction in RMSE, respectively, compared to GRU-TAE and MBS⁺ methods alone. This additional result encourages the use of such hybrid methods in future works, using a more complex integration.

5.5 Conclusion

The objective of this paper was to develop an approach for ball trajectory predictions in table tennis, combining both machine learning methods and physical models. To achieve this objective, a method for parameter identification of a ping-pong ball dynamic model using differentiable models was developed and combined with a GRU-TAE and a TAE.

First, the method evaluated the parameter identification process. Secondly, the accuracy of this method (MBS⁺), trajectory predictions for ping-pong balls, using optimized parameters was compared to GRU-TAE and TAE methods.

The main results are as follows: 1. Differentiable models enable satisfactory identification of dynamic parameters using only 50 ball trajectories. By using our custom ping-pong ball, the parameters identification resulted in a 42% reduction in prediction errors compared to the initial parameters from the literature. 2. On the real-world dataset, the MBS⁺ method achieved an RMSE

of 58 mm, which represents an 9% improvement compared to GRU-TAE. 3. The methods are applicable in real-time applications, even if a trade off can be done between accuracy and computation time by whether or not including spin estimation.

Finally, these results enhance robotic arm performance in returning balls to human players, allowing earlier positioning for ball return, and reducing late position adjustments. This work is aiming toward real-world deployment in collaborative table tennis robots.

Additionally, Hybrid^2 and Hybrid^3 methods, averaging both model-based and data-driven approaches were tested, further improving the accuracy of the predictions. Using the Hybrid^3 method, the RMSE on real trajectories was reduced to 49 mm, representing a further 16% improvement compared to MBS$^+$ methods. Future works will focus on optimizing the combination of these hybrid methods combining model-based and data-driven prediction by considering the uncertainties of each method and applying them to a real robotic ping-pong player system.

5.6 Statements and Declarations

Funding: Institute for data valorization (IVADO), Montreal, Canada.

Author contributions: Conceptualization: BT, MR Methodology: BT, MR. Investigation: BT, MR. Visualization: BT, MR. Funding acquisition: BT, MR. Project administration: BT, MR. Supervision: MR. Writing – original draft: BT. Writing – review & editing: BT, MR.

Competing interests: Authors declare that they have no competing interests.

5.7 Appendix

The TAE architecture consists of an encoder and a decoder composed by dense layers. The training and testing methodology was inspired by [11], [33]. To predict a trajectory consisting of m=150 positions, the inputs were a 150x4 matrix containing the known n positions of the ball along the x, y and z axes throughout the trajectory, along with a zero-padded mask to fill the matrix. This fixed length was used for all training and test data trajectories. The output of these models was a 150x3 matrix containing the x, y and z positions of the ball for the entire predicted trajectory as shown in Figure S1.

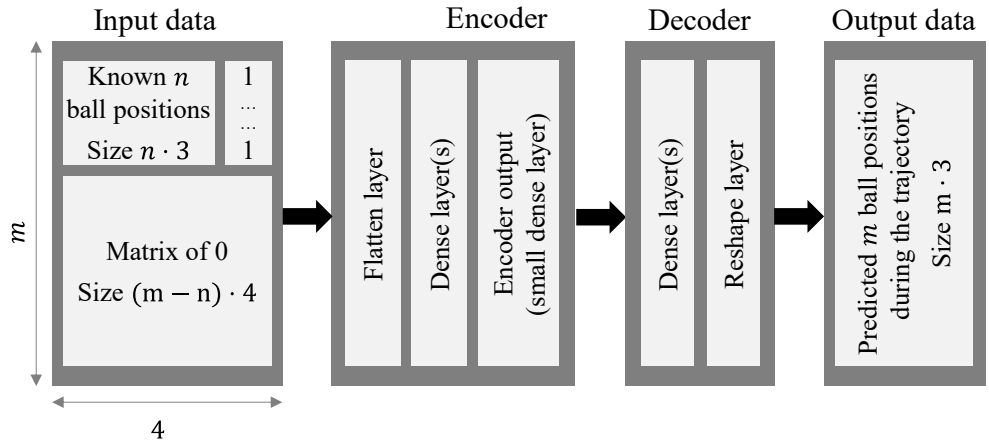


Figure S1. Architecture of the TAE for ping-pong ball trajectory prediction.

The architecture of our GRU-TAE is the same as [33] and is presented in Figure 2. The main difference between the TAE and the GRU-TAE is that the dense layers of the encoder are replaced by GRU layers. In [33], the performance of the architecture was evaluated, by testing prediction accuracy with different hyperparameters, including learning rate, batch size, the amount of training data and the size and number of layers. Detailed results for the optimal hyperparameters can be found in the Appendix of this previous study.

Interestingly, while traditional GRU-autoencoder designs replace the dense layers with GRU layers in both the encoder and the decoder, it has been shown that the GRU-TAE achieved better accuracy when GRU layers were used only in the encoder. In this approach, the GRU cells extract significant parameters from the trajectory, which are then used to reconstruct the trajectory (which is no longer a time series) using dense layers. The encoder of the GRU-TAE consists of two GRU layers: the first layer contains between 16 and 64 cells, and output a sequence, while the second layer is

another GRU layer with 16 cells. The final output of the encoder is a vector of 16 values. The decoder comprises two dense layers, one with up to 400 units and the other with 270 units.

All data-driven architectures were implemented using Tensorflow [35] and Keras libraries [38] in Python, and training was conducted on GoogleColab servers. To optimize computation time for the tested methods, the code was precompiled using the Numba library [39], allowing for faster execution. An open-source implementation of the code is available at [36].

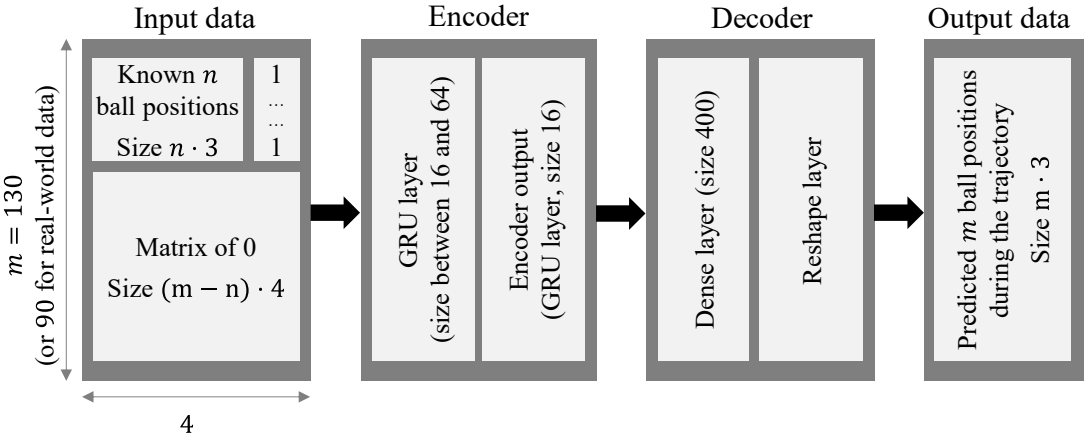


Figure S2. Architecture of the GRU-TAE for ping-pong ball trajectory prediction.

5.8 Références

[1] R. L. Anderson, *A robot ping-pong player: experiment in real-time intelligent control*. MIT press, 1988. Accessed: Jun. 19, 2024. [Online]. Available: https://dl.acm.org/doi/abs/10.5555/42334

[2] Y. Huang, B. Schölkopf, and J. Peters, "Learning optimal striking points for a ping-pong playing robot," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 4587–4592. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7354030

[3] J. Tebbe, Y. Gao, M. Sastre-Rienietz, and A. Zell, "A Table Tennis Robot System Using an Industrial KUKA Robot Arm," in *Pattern Recognition*, vol. 11269, T. Brox, A. Bruhn, and M. Fritz, Eds., in Lecture Notes in Computer Science, vol. 11269. , Cham: Springer International Publishing, 2019, pp. 33–45. doi: 10.1007/978-3-030-12939-2_3.

[4] L. Acosta, J. J. Rodrigo, J. A. Mendez, G. N. Marichal, and M. Sigut, "Ping-pong player prototype," *IEEE robotics & automation magazine*, vol. 10, no. 4, pp. 44–52, 2003.

[5] Y. Zhang, W. Wei, D. Yu, and C. Zhong, "A tracking and predicting scheme for ping pong robot," *Journal of Zhejiang University SCIENCE C*, vol. 12, no. 2, pp. 110–115, 2011.

[6] A. Kyohei, N. Masamune, and Y. Satoshi, "The ping pong robot to return a ball precisely," *Omron Technics*, vol. 51, pp. 1–6, 2020.

[7] Y. Zhao, R. Xiong, and Y. Zhang, “Rebound modeling of spinning ping-pong ball based on multiple visual measurements,” *IEEE Transactions on Instrumentation and Measurement*, vol. 65, no. 8, pp. 1836–1846, 2016.

[8] C. H. Lai and T. I. J. Tsay, “Self-Learning for a Humanoid Robotic Ping-Pong Player,” *Advanced Robotics*, vol. 25, no. 9–10, pp. 1183–1208, Jan. 2011, doi: 10.1163/016918611X574678.

[9] J. Tebbe, L. Klamt, Y. Gao, and A. Zell, “Spin detection in robotic table tennis,” in *IEEE international conference on robotics and automation (ICRA)*, 2020, pp. 9694–9700. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9196536/

[10] T. Gossard, J. Tebbe, A. Ziegler, and A. Zell, “SpinDOE: A ball spin estimation method for table tennis robot,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 5744–5750. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10342178

[11] S. Gomez-Gonzalez, S. Prokudin, B. Schölkopf, and J. Peters, “Real time trajectory prediction using deep conditional generative models,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 970–976, 2020.

[12] Y. Yang, D. Kim, and D. Choi, “Ball tracking and trajectory prediction system for tennis robots,” *Journal of Computational Design and Engineering*, vol. 10, no. 3, pp. 1176–1184, 2023.

[13] C. Zhang, X. Xi, X. Wang, and Z. Zhang, “Football trajectory prediction and real-time feedback mechanism based on Temporal Convolutional Network,” *Alexandria Engineering Journal*, vol. 114, pp. 476–484, 2025.

[14] Y. Gao, J. Tebbe, and A. Zell, “A model-free approach to stroke learning for robotic table tennis,” in *International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2022, pp. 1–8. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9892776/

[15] Y. Zhao, R. Xiong, and Y. Zhang, “Model based motion state estimation and trajectory prediction of spinning ball for ping-pong robots using expectation-maximization algorithm,” *Journal of Intelligent & Robotic Systems*, vol. 87, no. 3, pp. 407–423, 2017.

[16] K. Mülling, J. Kober, and J. Peters, “A biomimetic approach to robot table tennis,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 1921–1926. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5650305/

[17] K. M. Kulkarni and S. Shenoy, “Table tennis stroke recognition using two-dimensional human pose estimation,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 4576–4584. Accessed: Jun. 20, 2024. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2021W/CVSports/html/Kulkarni_Table_Tennis_Stroke_Recognition_Using_Two-Dimensional_Human_Pose_Estimation_CVPRW_2021_paper.html?ref=https://githubhelp.com

[18] TableTennisDaily, “World’s Best Table Tennis Robot vs TableTennisDaily’s Dan!” Accessed: Jun. 20, 2024. [Online]. Available: https://www.youtube.com/watch?v=u3L8vGMDYD8

[19] P. Blank, J. Hoßbach, D. Schuldhaus, and B. M. Eskofier, “Sensor-based stroke detection and stroke type classification in table tennis,” in *ACM International Symposium on Wearable Computers - ISWC ’15*, 2015, pp. 93–100. doi: 10.1145/2802083.2802087.

[20] P. Blank, B. H. Groh, and B. M. Eskofier, “Ball speed and spin estimation in table tennis using a racket-mounted inertial sensor,” in *ACM International Symposium on Wearable Computers*, Sep. 2017, pp. 2–9. doi: 10.1145/3123021.3123040.

[21] Y. Zhang, R. Xiong, Y. Zhao, and J. Wang, “Real-time spin estimation of ping-pong ball using its natural brand,” *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 8, pp. 2280–2290, 2015.

[22] T. Rémond, “Physique du rebond: application à la balle de tennis de table,” PhD Thesis, Ecole normale supérieure de lyon-ENS LYON, 2023. Accessed: Jun. 20, 2024. [Online]. Available: https://theses.hal.science/tel-04221365/

[23] R. H. Bao and T. X. Yu, “Collision and rebound of ping pong balls on a rigid target,” *Materials & Design*, vol. 87, pp. 278–286, 2015.

[24] R. Cross, “Impact behavior of hollow balls,” *American Journal of Physics*, vol. 82, no. 3, pp. 189–195, 2014.

[25] H.-I. Lin and Y.-C. Huang, “Ball trajectory tracking and prediction for a ping-pong robot,” in *9th International Conference on Information Science and Technology (ICIST)*, IEEE, 2019, pp. 222–227. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8836894/

[26] A. Van Den Oord, N. Kalchbrenner, and K. Kavukcuoglu, “Pixel recurrent neural networks,” in *International conference on machine learning*, PMLR, 2016, pp. 1747–1756. Accessed: Jun. 20, 2024. [Online]. Available: https://proceedings.mlr.press/v48/oord16.html

[27] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[28] A. Borovykh, S. Bohte, and C. W. Oosterlee, “Conditional Time Series Forecasting with Convolutional Neural Networks,” Sep. 17, 2018, *arXiv*. Accessed: Jun. 20, 2024. [Online]. Available: http://arxiv.org/abs/1703.04691

[29] Y. LeCun and Y. Bengio, “Convolutional networks for images, speech, and time series,” *The handbook of brain theory and neural networks*, vol. 3361, no. 10, 1995, Accessed: Jul. 09, 2024. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=e26cc4a1c717653f323715d751c8dea7461aa105

[30] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,” Dec. 11, 2014, *arXiv*. Accessed: Jul. 09, 2024. [Online]. Available: http://arxiv.org/abs/1412.3555

[31] A. Harsha, “The Ultimate Showdown: RNN vs LSTM vs GRU – Which is the Best? - Shiksha Online.” Accessed: Jun. 20, 2024. [Online]. Available: https://www.shiksha.com/online-courses/articles/rnn-vs-gru-vs-lstm/

[32] R. Yu, S. Zheng, A. Anandkumar, and Y. Yue, “Long-term Forecasting using Higher Order Tensor RNNs,” Aug. 23, 2019, *arXiv*. Accessed: Jun. 20, 2024. [Online]. Available: http://arxiv.org/abs/1711.00073

[33] B. Toussaint and M. Raison, “Real-time trajectory prediction of a ping-pong ball using a GRU-TAE,” *Appl Intell*, vol. 55, no. 5, p. 339, Apr. 2025, doi: 10.1007/s10489-024-06204-4.

[34] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” *Advances in neural information processing systems*, vol. 27, 2014, Accessed: Jul. 09, 2024. [Online]. Available: https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html

[35] M. Abadi *et al.*, “TensorFlow: Large-scale machine learning on heterogeneous systems.” Mountain View, CA: Tensorflow, 2015.

[36] B. Toussaint, *GitHub - LiBRTy-Lab/GRU-TAE_for_trajectory_prediction:* (2024). Accessed: Oct. 29, 2024. [Online]. Available: https://github.com/LiBRTy-Lab/GRU-TAE_for_trajectory_prediction

[37] A. Nakashima, Y. Kobayashi, Y. Ogawa, and Y. Hayakawa, "Modeling of rebound phenomenon between ball and racket rubber with spinning effect," in *ICCAS-SICE*, IEEE, 2009, pp. 2295–2300. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5335158/

[38] F. Chollet, "Building autoencoders in keras," *The Keras Blog*, vol. 14, 2016, Accessed: Oct. 29, 2024. [Online]. Available: https://www.academia.edu/download/56975530/Building_Autoencoders_in_Keras.pdf

[39] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: a LLVM-based Python JIT compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, Austin Texas: ACM, Nov. 2015, pp. 1–6. doi: 10.1145/2833157.2833162.

CHAPITRE 6 ARTICLE 3 : DESIGN OF A MINIMAL MODEL-FREE CONTROL STRUCTURE FOR FAST TRAJECTORY TRACKING OF ROBOTIC ARM

Published in *Applied Sciences*, on September 18, 2024.

Abstract: This paper designs a minimal neural network (NN)-based model-free control structure for the fast, accurate trajectory tracking of robotic arms, crucial for large movements, velocities, and accelerations. Trajectory tracking requires an accurate dynamic model or aggressive feedback. However, such models are hard to obtain due to nonlinearities and uncertainties, especially in low-cost, 3D-printed robotic arms. A recently proposed model-free architecture has used an NN for the dynamic compensation of a proportional derivative controller, but the minimal requirements and optimal conditions remain unclear, leading to overly complex architectures. This study aims to identify these requirements and design a minimal NN-based model-free control structure for trajectory tracking. Two architectures are compared, one NN per joint (INN) and one global NN (GNN), each tested on two serial robotic arms in simulations and real scenarios. The results show that the architecture reduces tracking errors (RMSE < 2°). The INN is accurate for decoupled joint dynamics and requires fewer training data than the GNN. A table summarizes the design process. Future works will apply this control structure to low-cost robotic arms and micro-movements.

Keywords: three-dimensional printing; low cost; machine learning; uncertainties

6.1 Introduction

The transfer of robotics into everyday life is currently accelerating, with 20 million robots expected to be in use worldwide by 2030 [1]. The ability of robotic arms to accurately follow specified trajectories is important in a large field of applications.

Control structure methods for the successful tracking of such trajectories by robotic arms represent a problem that has been studied for decades [2] and require either an accurate dynamic model or aggressive tracking with high-gain feedback. However, for most robotic arms, an accurate dynamic model can be difficult to obtain due to nonlinearities (e.g., friction) and uncertainties (e.g., inertial parameters) in these dynamic systems [3], making implementation on robotic arms difficult. Thus, the inability of these joint servo controllers to address these nonlinearities and uncertainties can lead to the degradation of accuracy in trajectory tracking [4]. Generally, in industrial robots, the

solution consists of finding a compromise between a reduction in the cycle time and an improvement in tracking accuracy [5]. This situation is currently accentuated by the attempts to develop 3D-printed and low-cost robotic arms, which can have higher friction and cheaper motors.

The most common control architecture consists of a joint torque control using a linear controller, such as a traditional proportional–integral–derivative (PID) controller. To compensate for the response time of the controller, a feedforward compensation using the dynamic model of the system is usually added. Widespread alternatives can use simplified models [6] or machine learning methods [3]. Notably, iterative methods, such as iterative learning control, are based on task repetition and allow for efficient trajectory tracking either with the addition of a dynamic model [5] or without any information on the dynamic model [7], [8]. However, if these methods are useful for repetitive tasks, such as working on assembly lines, the learned representation is not easily transferable between different trajectories.

Control structures based on neural networks (NNs) have attracted attention with their potential ability to generalize beyond a training set [5]. For trajectory tracking especially, the power of approximation of NNs has been used in control structures to learn either the forward or inverse dynamics of the system [9], [10], [11], [12]. According to [5], when using an NN architecture, the control problem falls either in reinforcement learning [13], [14], adaptive control [15], or optimal control [16].

To approximate a system's forward dynamics, several NN architectures, such as recurrent neural networks (RNNs), feedforward networks, or radial basis function (RBF) neural networks, have been used, e.g., [12], [17], [18], [19].In [20], NNs were used to learn inverse dynamic models, and they were included as a feedforward component for dynamic compensation. It has already been shown that NNs—especially RNNs and Long Short-Term Memory (LSTM) neural networks—can outperform other techniques, such as Gaussian Processes, to learn inverse dynamics when there are sufficient training data [21], [22]. However, most of these works operate at a torque-level control, which usually requires many parameters for the NNs—in the order of 10^5 in [23], for a 7-degrees of freedom (DOF) robot arm—or several neural networks to approximate different elements of the dynamic model. Moreover, when torque control is required, a bridge exists between the results in simulations and in reality. In [24], an adaptive controller based on a neural network was used for robot manipulator trajectory tracking, but solely the simulation results were presented.

Even if the design of a controller plays a major role in control structures, Ref. [25] reminds us that the design of the control structure also answers the following questions: “Which variables should be controlled and measured, and which inputs should be manipulated?” “Which control configurations (e.g., cascade control, feedforward control, etc.) should be used, how must the different controls be organized hierarchically, and how many degrees of freedom must the controller have to achieve the desired performance?”.

In [4], as in [26], another architecture is proposed for the control structure. An NN-based control structure is used in an outer loop (i.e., for position control) as feedforward compensation on either a quadrotor with a deep NN or a 7-DOF robot arm with an RNN. These papers highlighted three main advantages of combining an NN system as feedforward compensation with a position PD controller. First, NN architectures can be applied to various systems with complex dynamics while ensuring their stability [26]. Secondly, this approach is easy to use because it does not need any prior information on the system and can be applied to unseen trajectories without any adaptation process [5], [11]. Thirdly, with a good training process, this approach demonstrates good accuracy in trajectory tracking while being computationally efficient with a small model [4].

In [4], the learning of the direct dynamic model of a flexible Baxter robot was conducted at the joint position control level by combining a proportional–integral–derivative (PID) controller with either a bidirectional RNN or an RNN associated with an iterative learning control (ILC) algorithm, both with around 10^3 parameters, directly as one integrated model. The results showed a reduction in tracking errors between 12% and 54% depending on the joints, compared to the reference for random trajectories.

In [5], the inverse dynamic learning process of a 7-DOF industrial robot was first conducted in a simulation by implementing the control for many trajectories to collect data for the NN on the real system. However, this technique required a good dynamic simulation of the system, which is rarely available for non-industrial robots because a large amount of data is needed to train the model. In this paper, a trained NN with around 2.10^4 parameters—2 layers of 100 neurons—is employed as a feedforward component to compensate for the dynamic errors of the position controller of an ABB IRB6640-180 robot with 6 DOF. This approach resulted in a diminution of tracking errors by 80% to 90% compared to tracking errors without compensation, achieving a global mean squared error of 2.145° in their tests. Unlike their previous work [4], which used a single NN for the whole

system, the authors mentioned that on the studied ABB robot (Zurich, Switzerland), the joint dynamics were decoupled—i.e., the impact of the joint movements was negligible on the dynamics of the other joints—allowing the possibility to train six separate NN's, each approximating the dynamic inversion for one joint. However, only a global neural network for all joints was implemented.

In [11], we proposed a model-free control structure for high-speed trajectory tracking that combined the advantages of both proportional–derivative (PD) controllers, which are stable and easy to use, and neural networks, which are efficient in estimating the system dynamics. This method differed from [4] and [5] by using one NN per joint (the INN method) instead of one global NN for the entire robot arm (the GNN method). The architecture—illustrated in Figure 1—combines an NN with a PD controller. It consists of learning the dynamic response of the PD controller with an NN and using it in an outer loop as a feedforward compensation to predict and correct the errors of the PD controller, which is effective for robot trajectory tracking [27]. A summary of existing methods for trajectory tracking is presented in Figure 2.

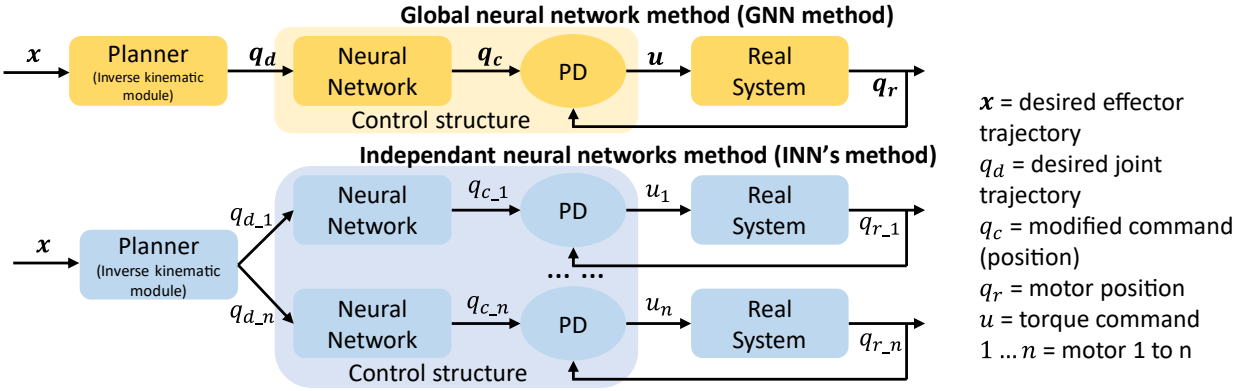


Figure 6.1 : Model-free control structure for high-speed trajectory tracking.

From our state-of-the-art review, it appears that using a control structure with an NN architecture as a feedforward component for the inner-loop dynamic compensation allows for reducing the errors in position and velocity during the trajectory tracking without any prior knowledge of the dynamic model [4,5,11]. However, there is a lack of studies optimizing the design process of this model-free control structure. This gap necessitates experimentally selecting learning parameters, often leading to unnecessarily complex architectures. Particularly, the current models still require a large number of parameters—ranging from 103 to 104 parameters—resulting in non-convex loss

functions and higher computational costs during the training process. The minimum number of parameters needed for the NN architecture to provide an effective dynamic compensation is also absent from the literature, as well as the amount of data required to train these architectures.

Moreover, there are no established guidelines on the best NN architecture for the use as feedforward compensation—whether a global NN for the whole system (the GNN method) or smaller NNs for each joint (the INN method)—and the necessary conditions to achieve optimal performances for these methods remain unclear.

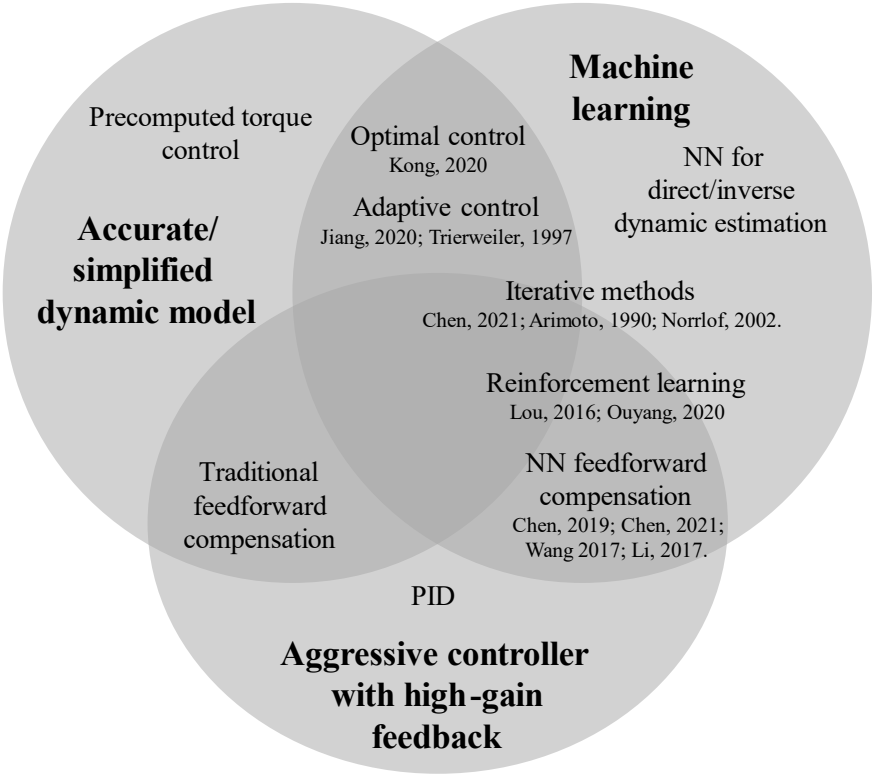
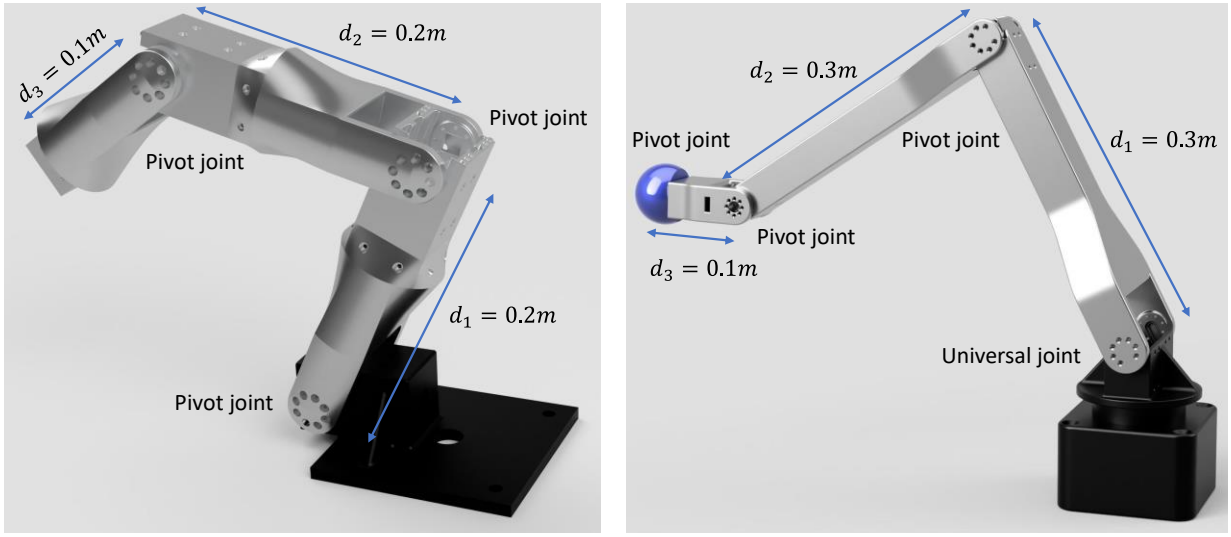


Figure 6.2 : Model-free control structure for high-speed trajectory tracking.

The objective of this study is to design a minimal model-free control structure based on a neural network for the fast and accurate trajectory tracking of robotic arms. The subsequent objective is to compare 1. one NN per individual joint (the INN method) and 2. one global NN for all joints (the GNN method). To meet this objective, both real 3D-printed robotic arms (3- and 5-DOF) and simulated robotic arms (3-DOF) were used. We focused on the fast trajectories, i.e., trajectories where the PD is unable to accurately track the trajectories due to the response time limitations. Figure 3 presents the design of the studied robots.

The planar 3-DOF setup allows straightforward results while accounting for external forces like gravity, while the 5-DOF setup demonstrates the architecture's applicability to a widely used topology in industry settings.



(**A**) (**B**)

Figure 6.3 : Design of the studied robotic arms for (A) simulated and real 3-DOF arms and (B) a real 5-DOF arm.

6.2 Materials and Methods

To obtain the required results to evaluate the NN architectures used in the control structure across different configurations, both real 3D-printed (3- and 5-DOF) and simulated robotic arms (3-DOF) were used, as shown in Figure 3.

First, the real 3-DOF planar robotic arm consisted of three 3D-printed bodies made from polylactic acid with segment lengths of $d_1 = 0.2$ m, $d_2 = 0.2$ m, $d_3 = 0.1$ m, respectively. Each body was connected by a pivot joint actuated by XM-540 Dynamixel motors.

Secondly, the real 5-DOF robotic arm consisted of five 3D-printed PLA bodies. Two DOFs were in the base (universal joint), while the remaining three were pivot joints, as shown in Figure 3. All joints were actuated by Dynamixel motors from the XM series. The segment lengths were $d_1 = 0.3$ m, $d_2 = 0.3$ m, $d_3 = 0.1$ m, respectively.

The experimental results were used to validate and enhance the simulated results, ensuring that the simulations represented reality. Specifically, we adjusted simulation parameters, including PID

gains, inertias, and masses, to ensure that the accuracy of the simulated PD controller matched that of the real PD controller.

We also assumed that the internal dynamics of the robotic arms were deterministic and repeatable.

6.2.1 Equations of the Simulated Model

The equations for the simulated system were generated using Robotran software (version 1.22.0) [28]. The semi-explicit formalism, as presented in (1), also referred to as the direct dynamic model in the literature, was used as follows:

$$\boldsymbol{M}(\boldsymbol{q},\boldsymbol{\delta})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q},\dot{\boldsymbol{q}},\boldsymbol{\delta},\boldsymbol{frc},\boldsymbol{trq},\boldsymbol{g}) = \boldsymbol{\tau}(\boldsymbol{q},\dot{\boldsymbol{q}})$$

where $\boldsymbol{M}[n \cdot n]$ is the symmetric generalized mass matrix of the system; $\boldsymbol{c}[n \cdot 1]$ is the nonlinear dynamic vector containing the gyroscopic, centrifugal and gravity terms, as well as the contributions of external forces and torques; $\boldsymbol{q}[n \cdot 1]$ contains the relative generalized coordinates; $\boldsymbol{\delta}[10n \cdot 1]$ contains the dynamic parameters of the system, i.e., masses, centers of masses, inertia; and $\boldsymbol{\tau}[n \cdot 1]$ contains the generalized joint forces.

To be as realistic as possible, the dynamic effects of the motors were considered by accounting for the inertia of the rotor of each motor and viscous friction, which is velocity-dependent [29]. For simplicity, we neglected the nonlinear friction torque given in this article. Thus, the model of the motors was written as follows:

$$\boldsymbol{J_m}\ddot{\boldsymbol{q}}_{\boldsymbol{m}} + \boldsymbol{D_m}\dot{\boldsymbol{q}}_{\boldsymbol{m}} = \boldsymbol{\tau_m} - \boldsymbol{R}\boldsymbol{\tau}$$

with $\boldsymbol{\tau}$ corresponding to the generalized joint forces from Equation (1); $\dot{\boldsymbol{q}}_{\boldsymbol{m}}[n \cdot 1]$ and $\ddot{\boldsymbol{q}}_{\boldsymbol{m}}[n \cdot 1]$ being the velocities and accelerations of the n motors, respectively; $\boldsymbol{J_m}$, $\boldsymbol{D_m}$, and $\boldsymbol{R}$ being three $[n \cdot n]$ diagonal matrices corresponding to the moments of inertia, the viscous friction coefficients of the motors, and the gear reduction ratios, respectively.

Finally, $\boldsymbol{\tau_m}[n \cdot n]$ is the vector of torques supplied by the motors, which corresponds to the output of the PD controller. Based on [29], we used (1) to replace $\boldsymbol{\tau}$ in (2) and applied the relation $\boldsymbol{q} = \boldsymbol{R}\boldsymbol{q_m}$ to combine the equations to obtain the complete model of the simulated arm:

$$\boldsymbol{J_m}\ddot{\boldsymbol{q}}_{\boldsymbol{m}} + \boldsymbol{D_m}\dot{\boldsymbol{q}}_{\boldsymbol{m}} = \boldsymbol{\tau_m} - \boldsymbol{R}(\boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{c})$$

$$(\boldsymbol{J_m} + \boldsymbol{RMR})\ddot{\boldsymbol{q}}_{\boldsymbol{m}} + \boldsymbol{D_m}\dot{\boldsymbol{q}}_{\boldsymbol{m}} + \boldsymbol{Rc} = \boldsymbol{\tau_m}$$

$$\ddot{\boldsymbol{q}} = \boldsymbol{R}\ddot{\boldsymbol{q}}_{\boldsymbol{m}} = \boldsymbol{R}(\boldsymbol{J}_{\boldsymbol{m}} + \boldsymbol{RMR})^{-1}(\boldsymbol{\tau}_{\boldsymbol{m}} - \boldsymbol{D}_{\boldsymbol{m}}\,\dot{\boldsymbol{q}}_{\boldsymbol{m}} - \boldsymbol{Rc})$$

An n-DOF robot arm was considered with a joint servocontrol with input $\boldsymbol{q_c} \in R^n$, the joint position commands, and the output $\boldsymbol{q} \in R^n$, the measured joint positions. Our goal was to find a feedforward compensation such that, for desired joint trajectories $\boldsymbol{q_d}$, the corrected input commands $\boldsymbol{q_c}$ would result in the perfect tracking of these desired trajectories: $\boldsymbol{q_c} \rightarrow \boldsymbol{q} = \boldsymbol{q_d}$. The architecture of the control structure is presented in Figure 1.

6.2.2 Implementation of the Control System

Once the gains of the PD controller were tuned by using traditional tuning methods, the NN architecture for feedforward compensation was designed. All NN architectures were built by using a multi-layer perceptron regressor, implemented with the "MLPRegressor" function from the Python library "sklearn" [30]. We tested architectures with varying numbers of neurons, starting from the simplest with a single layer of one neuron to more complex configurations, to determine the minimal requirements for achieving good accuracy. In simulation, we evaluated different motor reduction ratios and noise magnitudes to analyze their impact on the control structure accuracy. The range parameter values were set to reflect realistic values of physical systems.

Table 1 sums up parameters tested across different configurations. We provided an open-source implementation and training procedure for the architectures discussed in this section in [31].

Tableau 6.1 :Summary of tested parameters.

Parameter	Values
Arm	Real (3/5 DOF), Simulated (5 DOF)
NN Architecture	INN, GNN
Number of neurons	1, 3, 16, 3 + 3, 16 + 16, 3 × 16
Additional parameters studied in simulation	
PD parameters	RNN
Motor reduction ratio	1, 0.5, 0.25, 0.16, 0.1, 0.01
Noise magnitude (m)	0, 0.001, 0.002, 0.003, 0.005, 0.01, 0.02, 0.04

6.2.3 Test Procedure

The main steps for designing an accurate control structure with a feedforward neural network were identified prior to the study. To provide a guide for the design of our minimal model-free control

structure by using neural networks for feedforward compensation, we studied each of these steps in detail.

Using both real robotic arms and simulated models, we followed these steps:

Build a dataset of trajectories, all with the same duration representing the entire workspace of the arm, by using a simple PD controller on each joint, with the following control law.

$$\boldsymbol{c}(t) = \mathrm{P}\big(\mathbf{q}(\mathrm{t}) - \mathbf{u}(\mathrm{t})\big) + \mathrm{D}(\dot{\mathbf{q}}(\mathrm{t}) - \dot{\mathbf{u}}(\mathrm{t}))$$

with $\boldsymbol{c}$ the vector of commands (in current) of the n motors, $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ the position and velocity of the n motors, $\boldsymbol{u}$ and $\dot{\boldsymbol{u}}$ are the desired position and velocity of the n motors, and P and D the gains of the internal PD controller.

The dataset comprised 1000 trajectories, a number selected to provide enough data to ensure optimal training while being feasible for real systems.

Joint trajectories were generated from a home position by using quintic polynomials by choosing the desired random position and velocity in the workspace of each motor, as follows:

$$q_i(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$

With

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & t_d & t_d^2 & t_d^3 & t_d^4 & t_d^5 \\ 0 & 1 & 2t_d & 3t_d^2 & 4t_d^3 & 5t_d^4 \\ 0 & 0 & 2 & 6t_d & 12t_d^2 & 20t_d^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} q_{i0} \\ \dot{q}_{i0} \\ \ddot{q}_{i0} \\ q_{id} \\ \dot{q}_{id} \\ \ddot{q}_{id} \end{bmatrix}$$

with q_{i0} being the home position of the i[th] motor, $\dot{q_{i0}} = 0$ and $\ddot{q_{i0}} = 0$ being the initial velocity and acceleration of the motor, q_{id}, $\dot{q_{id}}$ being the desired position and velocity, and $\ddot{q_{id}} = 0$ being the desired acceleration.

The same equations were used to generate the second half of the trajectory between q_{id} and $q_{if} = q_{i0}$.

All these trajectories were generated at 100 Hz to ensure regular commands were sent to the motors with our computer.

Once the trajectories were executed, the dataset was built using a desired number of trajectories (between 20 and 1000). Each datum of the dataset consisted of five variables for each motor: the position $\boldsymbol{q}$ and the velocity $\dot{\boldsymbol{q}}$ of the motor at two consecutive timestamps—$\boldsymbol{q}(t)$, $\boldsymbol{q}(t+\delta)$, $\dot{\boldsymbol{q}}(t)$, $\dot{\boldsymbol{q}}(t+\delta)$—and the command sent between these two timestamps $\boldsymbol{u}(t)$.

Train the NN architecture to learn the dynamic model through the response of the PD controller. By giving the NN a lot of data from the trajectories, it learned which command had to be sent to each motor to go from a state—$\boldsymbol{q}(t)$, $\dot{\boldsymbol{q}}(t)$—to the next one—$\boldsymbol{q}(t+\delta)$, $\dot{\boldsymbol{q}}(t+\delta)$—and implicitly, the dynamic model of the robotic arm. The multi-layer perceptron regressor was trained with a maximum of 500 iterations by using the "MLPRegressor.fit ()" function from the Python library "sklearn" [30]. This function used the square error as the loss function:

$$L = \sum(\boldsymbol{u}(t) - \boldsymbol{q}(t+\delta)^2)$$

The loss function was minimized by using the Adam solver.

Generate a new set of commands to effectively track the desired trajectory $\boldsymbol{q_d}(t), t \in [0, t_f]$. For the GNN method, inputs consisted of the ensemble of two successive positions and velocities for all motors—$\boldsymbol{q}(t)$, $\dot{\boldsymbol{q}}(t)$, $\boldsymbol{q}(t+\delta)$, $\dot{\boldsymbol{q}}(t+\delta)$. For the INN method, each NN received inputs related solely to its specific motor, i.e., its successive position and speed—$q_i(t)$, $\dot{q}_i(t)$, $q_i(t+\delta)$, $\dot{q}_\iota(t+\delta)$.

Outputs were the series of commands required for each motor, by considering the response time of the PD controller to ensure the execution of the desired trajectory. Table 2 summarizes how a command sequence was computed for the trajectory of the joint *i*.

Tableau 6.2 : Computation of a command sequence.

Step	Inputs	Output
1	$q_i(0)$, $\dot{q}_i(0)$, $q_i(\delta)$, $\dot{q}_i(\delta)$	$u_i(t)$ for $t \in [0;\ \delta]$
2	$q_i(\delta)$, $\dot{q}_i(\delta)$, $q_i(2\delta)$, $\dot{q}_i(2\delta)$	$u_i(t)$ for $t \in [\delta;\ 2\delta]$
3	$q_i(2\delta)$, $\dot{q}_i(2\delta)$, $q_i(3\delta)$, $\dot{q}_i(3\delta)$	$u_i(t)$ for $t \in [2\delta;\ 3\delta]$
n	$q_i((n-1)\,\delta)$, $\dot{q}_i((n-1)\,\delta)$, $q_i(n\,\delta)$, $\dot{q}_\iota(n\,\delta)$	$u_i(t)$ for $t \in [(n-1)\delta;\ n\delta]$

Store the tracking errors for n test trajectories ($n = 40$). The number of test trajectories was chosen to ensure low variation in the results while maintaining a reduced number of trajectories for practical feasibility. The accuracy of the control structure for trajectory tracking was evaluated by

comparing the root-mean-square errors (RMSE) along the trajectory, $e = \sqrt{mean(\sum(\boldsymbol{q} - \boldsymbol{q_d})^2)}$, with $\boldsymbol{q}$ being the real executed trajectory and $\boldsymbol{q_d}$ being the desired trajectory.

This test procedure was applied by using different values of parameters and NN architectures to compare their effectiveness to track the test trajectories accurately.

6.3 Results

Figures 4 and 5 illustrate the impact of the motor reduction ratio on the performance of the control structure for both the GNN and INN methods, which must be considered during the design phase to choose the best architecture for the NN. Additionally, these figures present performances for varying amounts of training trajectories, which is important for optimizing the NN training process.

6.3.1 Results with the 3-DOF Simulated Model without Reduction Ratio

Figure 4A illustrates a typical learning curve obtained with the simulated model from Figure 3A by using both methods. One can observe that the INN method requires 100 trajectories, while the GNN method needs 250 trajectories of 1 second each to maximize their accuracy.

The learning curves for motors 1 and 2 highlight the differences in tracking accuracy between the two studied architectures when more than 100 trajectories are available. The INN method achieves an RMSE of 0.2 radians, whereas the GNN method achieves an RMSE of less than 0.04 radians—or 2.3 degrees. Conversely, the learning curve for the third motor shows an RMSE of 0.016 radians—less than 1 degree—for both methods, representing a 91% reduction in tracking errors compared to the PD controller without feedforward compensation.

6.3.2 Results with the 3-DOF Simulated Model with a Reduction Ratio

Figure 4B presents the learning curves for the two architectures with a reduction ratio R of 0.01. This time, RMSE is under 0.05 radians—3 degrees—for all three motors.

As observed in Figure 4A, the INN method reaches its optimal performance with 100 trajectories vs. 250 for the GNN method.

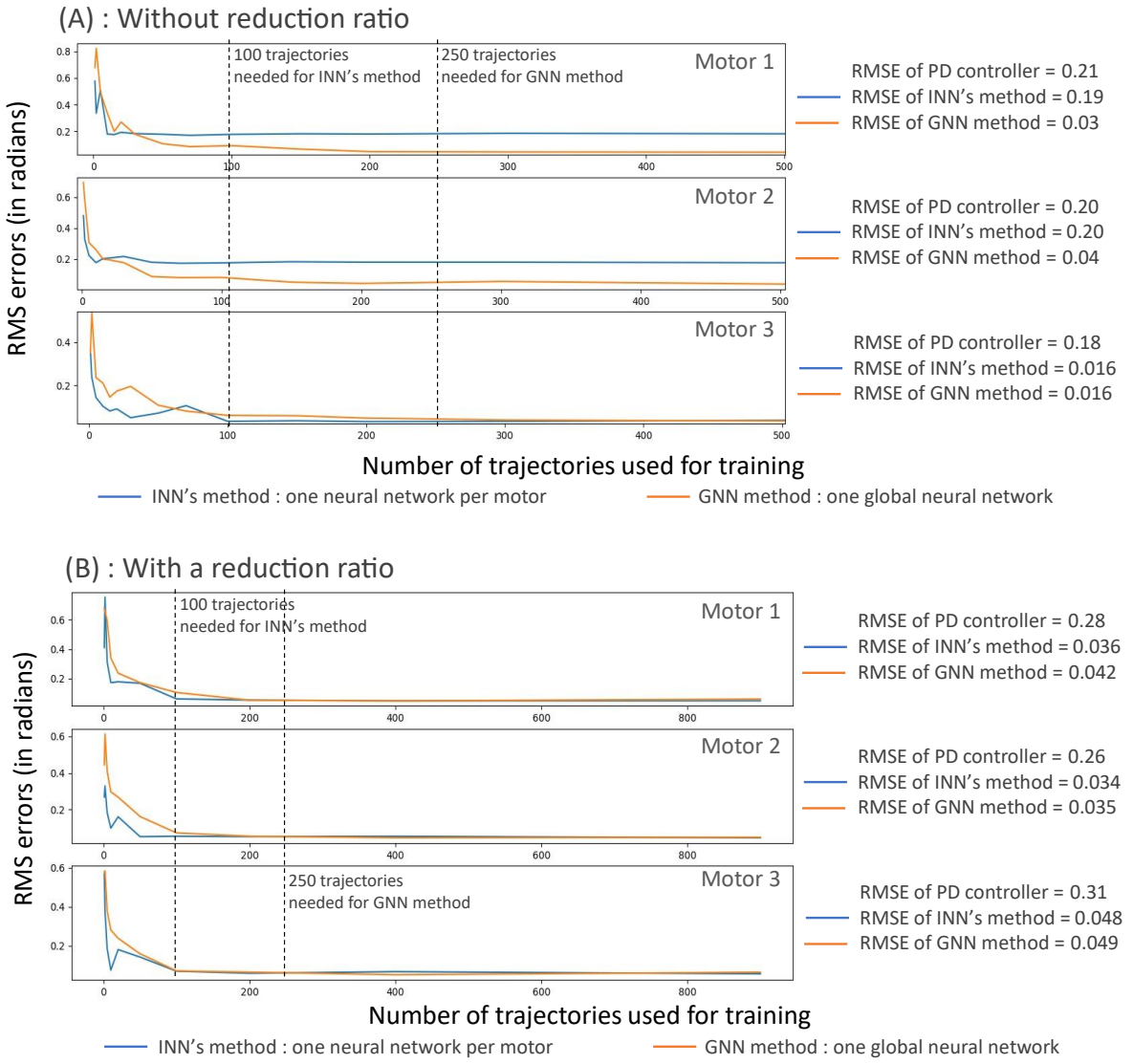


Figure 6.4 : Learning curve with both INN and GNN methods: (A) without reduction ratio, (B) with a reduction ratio R = 0.01.

Figure 5 presents the average relative performance of both methods, with the reference PD controller as a function of the ratio between the inertia $\boldsymbol{J_m}$ of the motor rotor and the mass matrix of the system ($\boldsymbol{RMR}$). The results indicate similar performances for both methods for a ratio above 10 with a relative RMSE of 0.15 compared to the PD controller and a progressive deterioration of performances of the INN method when the ratio decreases. For a ratio of 0.2—obtained when there

is no reduction ratio in the motor—the GNN method demonstrates a relative RMSE of 0.22 compared to 0.56 for the PD controller, i.e., 2.5 times less.

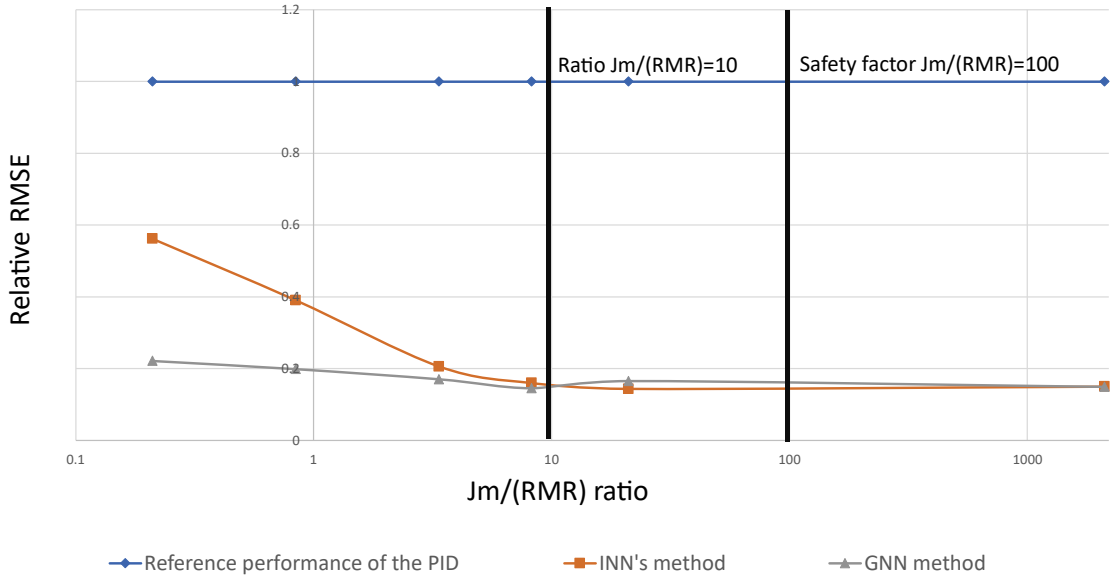


Figure 6.5 : Relative performances of the NN-based control structure compared to the reference PD controller as a function of the $\boldsymbol{J_m/(RMR)}$ ratio.

Figure 6 shows the relative performances of both methods compared to the reference PD controller depending on its accuracy—or parameters—for two ratios between the frequency of the dynamic integration and the frequency of the feedforward NN compensation to construct an optimal dataset. When the step time corresponds to the interval between two commands from the neural network, the RMSE of the NN is nearly proportional to that of the PD controller, showing a reduction of 90% of its tracking errors. When the frequency of the dynamic integration is ten times higher than the frequency of the NN, the reduction in tracking errors compared to the PD controller is between 80% and 85%, resulting in RMSE values that are two times more important than when the timestamps are the same. Moreover, when the reference PD is more aggressive—i.e., manages to track the trajectories with an RMSE less than 3 degrees—we can see that the performances of the NN are no longer proportional to that of the PD with an RMSE remaining around 0.5 degrees when the two frequencies differ.

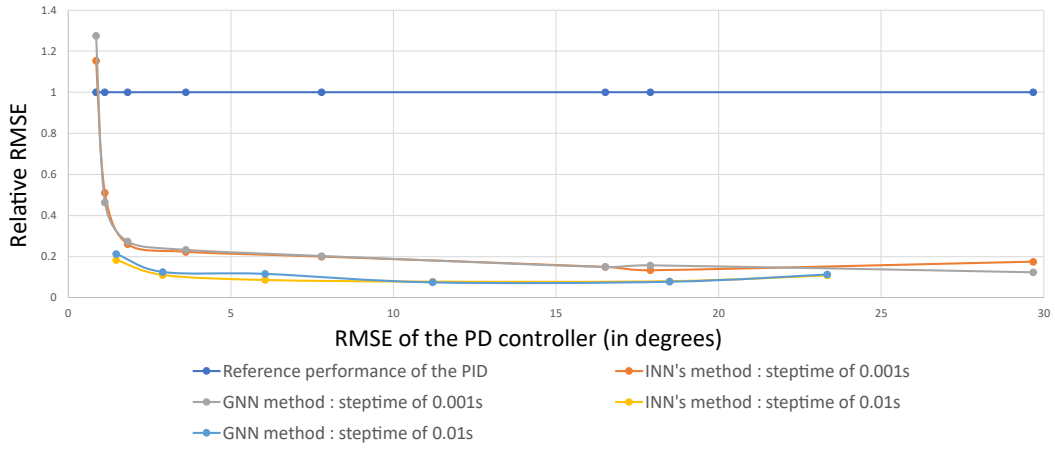


Figure 6.6 : Relative performances of the NN-based control structure compared to the reference PD controller as a function of the accuracy of the PD controller and the step time of the direct dynamic integration.

To design an optimal neural network, the median relative RMSE was compared for different sizes of NNs for both INN and GNN methods tested under various PID parameters. The results are presented in Table 3. All tested architectures, which have at least as many neurons as the number of associated motors on each layer, achieved a median relative accuracy between 0.935 and 1.063 compared to the smallest neural network, representing less than a 10% difference in accuracy. On the contrary, when the number of neurons per layer was fewer than the number of associated motors, the NN failed to converge. This is true for the NN with only 1 neuron in the GNN method, which had an RMSE 24.26 times greater than other NN configurations.

Tableau 6.3 : Median relative RMSE according to the size of the NNs.

Method	One NN for Each Servomotor						One global NN for the Whole System					
Size of the NN	1	3	16	2 × 3	2 × 16	3 × 16	1	3	16	2 × 3	2 × 16	3 × 16
Relative RMSE	1	1.057	0.978	0.935	0.992	0.996	24.26	0.970	1.014	1.027	0.989	1.063

Figure 7 shows the relative performances of both methods compared to the reference PD controller depending on the noise present in the training data. In the absence of noise, both INN and GNN

methods achieved a 90% improvement in trajectory tracking over the reference PD controller, with an RMS error of 0.86 degrees. A gradual decrease in the accuracy of the system is observed for noise between 0 and 0.5 degrees and is also observed for both INN and GNN methods.

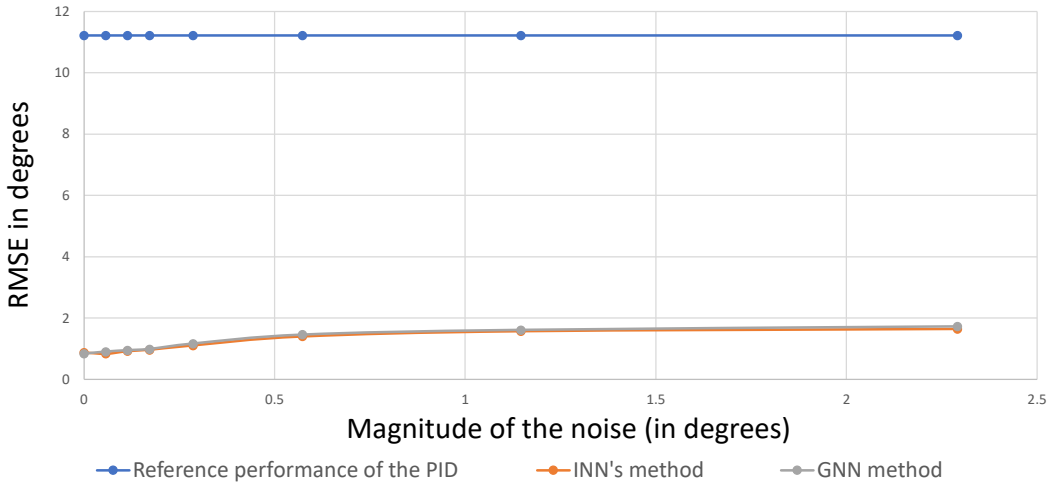


Figure 6.7 : Relative performances of the neural networks to the reference PD controller as a function of the noise magnitude in the training data.

Moreover, when the noise exceeds 0.5 degrees, the RMSE becomes 1.8 times larger than in the absence of noise in the data. However, the RMSE of the NN remains relatively stable—ranging between 1.40 and 1.65 degrees—which is still five times less than the RMSE of the PD controller.

Note that the noise in this figure is quite high; for instance, the noise in the real systems in this study is estimated to be 0.2 degrees (one encoder step).

Finally, with noise levels higher than 0.5 degrees, the NNs with more neurons demonstrated a 20% improvement in accuracy compared to the smallest tested NNs, as summarized in Table 3.

6.3.3 Experimental Results

Figure 8 presents a comparison between trajectory tracking accuracies—measured by RMSE—of the control structure for three configurations using real robotic arms between the two studied NN architectures: (A) fast trajectories and (B) intermediate trajectories for a 3-DOF robotic arm and (C) slow trajectories with a 5-DOF serial robotic arm.

Figure 8 also shows that the INN method achieves accuracy close to that obtained with all training data by using solely 100 trajectories, whereas the GNN method requires around 300 trajectories to effectively learn the dynamic response of the PD controller, regardless of the robot arm or trajectory speed.

Table 4 summarizes the numerical RMSE values among the n-test trajectories and the reduction rates of the errors compared to the reference PD controller. Specifically, the INN method consistently shows at least a 75% reduction in tracking errors compared to the PD controller alone, with an RMSE under 1 degree across all configurations.

Tableau 6.4 : Comparison between the trajectory tracking accuracies (RMSE) of NNs for trajectory tracking on two real robot arms.

Trajectory Velocities (and DOF)	RMSE with the Reference PD Controller	RMSE with the GNN Method	RMSE with the INN Method
	In Degrees (Tracking Errors Prediction Compared to the Reference Controller in %)		
Fast (3 DOF)	5.36	1.12 (79%)	0.88 (84%)
Intermediate (3 DOF)	8.17	0.45 (94%)	0.33 (96%)
Slow (5 DOF)	3.18	1.71 (46%)	0.76 (76%)

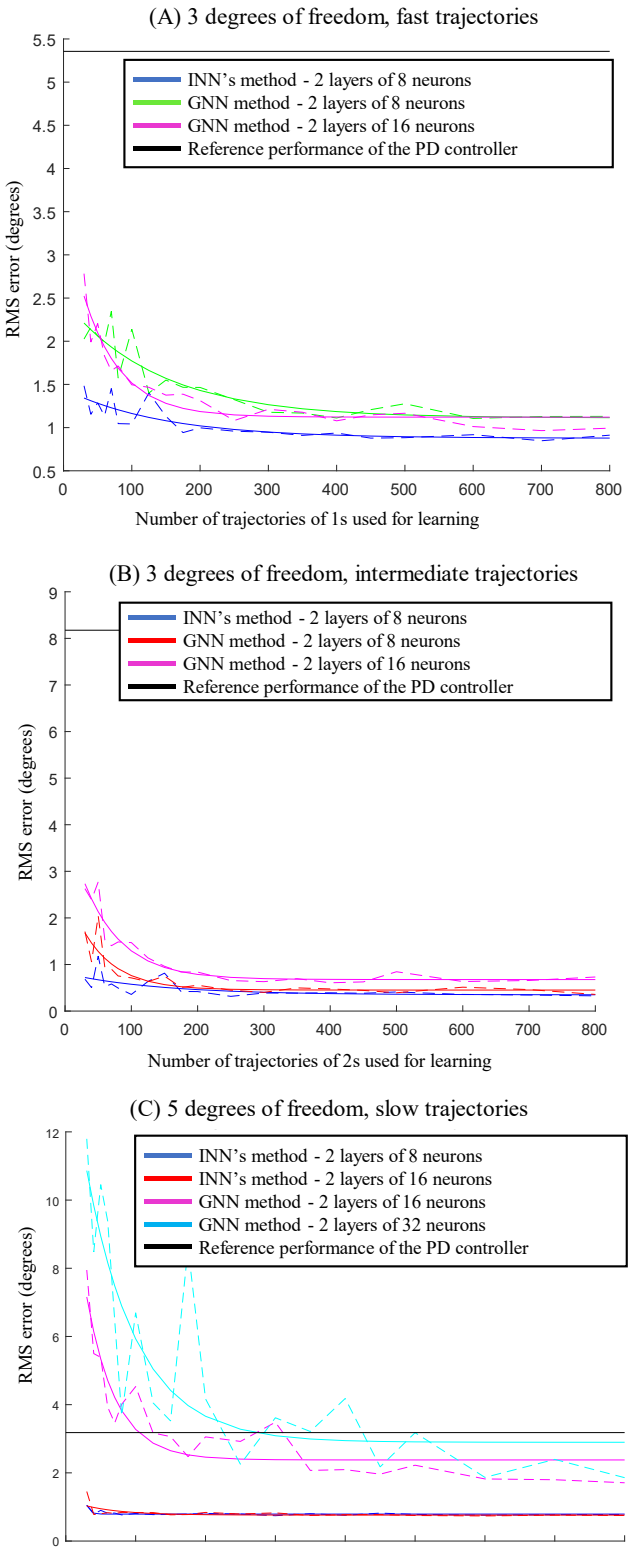


Figure 6.8 : Performance of trajectory tracking with various NNs for (A) 3-DOF robotic arm, fast trajectories (B) 3-DOF robotic arm, intermediate trajectories; (C) 5-DOF robotic arm, slow trajectories.

6.4 Discussion

The first main result is highlighted by the simulated model without reduction ratio in Figure 4A. The INN method is effective solely when the joint dynamics are not coupled with the other joints' dynamics—i.e., the impact of the joint movements on the dynamics of the other joints is negligible. In fact, the INN method performs well for the last motor, with a 91% reduction in tracking errors. This is because the third motor at the end of the arm is not affected by the movements of the first two motors. Contrarily, the INN method does not outperform the PD controller for the first two motors, with an RMSE of 0.2 radians similar to that of the PD controller.

The second main result is highlighted in Figure 4B: the INN method outperforms the GNN method when the dynamics of each joint are not coupled with the other ones and when the training dataset comprises fewer than 250 trajectories. This can be explained by the lower number of parameters required for the INN method. Moreover, both methods present a similar accuracy when more training data are available, with RMSE values ranging from 0.034 and 0.048 for the INN method and from 0.035 and 0.049 for the GNN method. These first two results align with those in [4], where the GNN method was applied to a flexible Baxter robot because of the dependence on the joint dynamics and the statement that it would be possible to use the INN method on an ABB industrial robot because of the decoupled dynamics [4], [5].

The decoupling of the joint dynamics occurs when the term $\boldsymbol{RMR}$ in Equation (2) becomes negligible compared to the inertia of the servomotor $\boldsymbol{J_m}$. This means that the influence of the mass matrix on the system is negligible compared to the inertia of the servomotor due to the reduction ratio. This result is consistent with the equations presented in [29]. Specifically, Figure 5 shows that this decoupling occurs when the inertia of the rotor of the motor $\boldsymbol{J_m}$ is more than 10 times larger than the term $\boldsymbol{RMR}$ in Equation (2). In this case, the INN method requires less data than the GNN method to complete the learning process: 100 trajectories vs. 250 trajectories for a 3-DOF robot arm. Moreover, the more complex the controlled robot arm, the more trajectories required for the training of the NN in the GNN method. To our knowledge, no study has explicitly demonstrated this result. Figure 3 also shows that the more trajectories available for training, the more accurate will be the control structure. Notably, the amount of data required for the training is 50 times less than the 500 trajectories of 25 s used in [4]. These ones likely used more trajectories than necessary to ensure an optimal training process of their RNN.

Contrarily, Figures 4A and 5 also show the advantages of using the GNN method for systems with coupled dynamics, i.e., when the inertia of the rotor of the motor $\boldsymbol{J_m}$ is no more than 10 times larger than $\boldsymbol{RMR}$ in Equation (2). In this case, the GNN method provides better accuracy. For example, when the reduction ratio is 1, the RMSE is 2.5 times smaller than the INN method. In fact, the global NN in the GNN method can learn the dependencies between the different joints and maintain accuracy close to that achieved with decoupled dynamics. To our knowledge, no study has explicitly compared the INN and GNN approaches in systems with coupled dynamics.

To summarize, these first results provide guidelines for both designing the neural network system and its training: if the ratio is greater than 10, a small neural network per servomotor can be used effectively. If the ratio is less than 10, a larger neural network for the entire system will yield better accuracy. Moreover, when the $\boldsymbol{J_m/(RMR)}$ ratio is between 10 and 100, the accuracy of the GNN method might be slightly better than the accuracy of the INN method. Thus, if the system accuracy is crucial, it may be beneficial to use both methods to see if the improvement in accuracy justifies the use of the GNN method.

Further, Figure 6 demonstrates that to achieve the best possible accuracy for both the INN and GNN methods, the step time of the training data must match the PD frequency used during the dataset construction. In fact, when the step time of the data used for training corresponds to the PD frequency, the RMSE is twice as low compared to scenarios where the step time is 10 times smaller than the PD frequency. However, to enable the neural network to effectively learn the dynamics of the PD controller, sufficient resolution in the data is required. Particularly, the motor should be able to complete at least 20 encoder steps between 2 consecutive time steps. Thus, the resolution of the encoders can limit the step time of the dataset. To our knowledge, none of the existing literature addresses the impact of the PD frequency on the tracking accuracy and the requirements for encoder resolution for this control structure.

Figure 6 also shows that the more accurate the PD controller, the more accurate the NN-based control structure. Thus, it is strongly recommended to optimize the gain of the PD controller before starting the training process, to ensure the best accuracy for the control structure without the feedforward neural network. In fact, the relative accuracy of the NN-based control structure compared to the PD controller alone remains stable regardless of the accuracy of the PD controller.

The tuning can be performed by using manufacturer recommendations when available or through classical tuning methods from the literature.

Table 3 indicates that a necessary condition for the successful use of the GNN method is that the number of neurons in each layer should be larger than or equal to the number of motors in the system for proper convergence. When this condition is not met, the global NN is forced to express the dynamics of multiple motors in one neuron, which is insufficient to accurately represent the joint dynamics. To our knowledge, the current study is the first one to provide guidelines on the minimal size required for the NN architecture to efficiently learn dynamic compensation.

Furthermore, Table 3 shows that, theoretically, when the INN method is used, each neural network may require only a single neuron. In fact, the RMSE comparison for trajectory tracking has shown that when this condition is met, there is no significant difference in accuracy—less than 10 percent—regardless of the NN size. This aligns with the results from [5], which tested various NN sizes, showing no significant performance differences. The best accuracy was achieved with one of the smallest tested architectures. This result encourages the use of minimal neural networks to limit the number of parameters and improve the convergence process.

Compared to other NN-based control structures in the literature, the smallest NN presented in Figure 4B achieves an accuracy comparable to larger networks while requiring only 15 parameters for the entire 3-DOF robot arm—5 parameters per motor. This is 67 times fewer than the system described in [4] and 6700 times fewer than the architecture in [23] that operates at the torque level. This allows a proportional diminution of the calculation time, allowing for increased frequency of the control structure and performance.

In the same way, when the GNN method is used, the NN should obtain at least as many neurons as there are joints in the system (on each layer). Note that in real systems, it can be necessary to repeat the training process of the NN to achieve convergence, so it is possible to gradually increase the number of neurons and the number of layers until having a good convergence.

Finally, Figure 7 shows that while not critical, it is good practice to use the highest possible encoder resolution to limit the noise in the data. Although a significant noise—more than 0.5 degrees—does not prevent the control structure from improving the trajectory tracking, with an RMSE ranging from 1.4 to 1.65 degrees (five times lower than the PD controller), lower noise levels yield better results. Even though the NN seems capable of managing this noise during the training, Figure

7 also shows that the best results are obtained when the noise is zero—or almost zero with an RMSE of 0.86, 1.8 times lower than when the noise exceeds 0.5 degrees.

Figure 8 and Table 4 reiterate the interest in using an NN as a feedforward compensation in the control structure, applying both the INN and GNN methods to a real 3D-printed robotic arm. Both methods allow an RMSE of less than 2 degrees for all tested configurations during the trajectory tracking on the real physical robot arms. These results are consistent with the errors observed in the simulated model and the errors of the system in [4], i.e., over a 50% improvement for multi-joint trajectories and 40% for random joint trajectories. In fact, in Table 4, the use of the INN method demonstrates a 76% to 96% reduction in errors across all tested configurations. This result matches the simulation results of Figures 3B and 4 with an 80% to 90% reduction in tracking errors and is better than the 40% improvement for random joint trajectories in [4]. This difference is likely due to the type of robot used—a flexible Baxter robot. In [5], where an industrial ABB robot is used, an 80% to 90% reduction has also been reported compared to the reference PD controller alone. Note that the aim of the INN and the GNN methods is not to outperform the accuracy of control structures that include accurate dynamic models when available but rather to offer a simple and reliable way to accurately track desired trajectories.

Specifically, Figures 8A and 8B show that the improvement is even more significant for faster trajectories, where the PD controller alone is less accurate than slower trajectories, with a 96% and an 84% reduction for the fastest speed vs. a 76% reduction for the slowest trajectories. Significantly, the INN method achieves an RMSE under 1 degree across all configurations. This result is better than the accuracy of all control structures tested in [5], which had RMSE values between 2.14 and 2.62 degrees, while also using a smaller number of parameters in the NN architectures—1 neuron per motor vs. at least two layers of 50 neurons.

The experimental results in Figure 8 also show that the GNN method does not outperform the accuracy of the INN method. These results are consistent with the decoupled joint dynamics, i.e., the dynamics of the servomotors prevail over the dynamics of the robotic arm due to the real reduction ratio of the servomotors (1/540).

Finally, the experimental results confirm the main results from the simulated model on a real system: the INN method requires two to three times less data than the GNN method to effectively learn the dynamics.

A guide summarizing the conclusions of this study for designing a neural network-based control structure for position control is presented in Table 5.

Tableau 6.5 : Guide for the design of a neural network-based control structure for position control, followed by an example of its application on the real 3- and 5-DOF systems.

Step 1—Tune the Gains of the PD Controller. Tune the gain values of the PD controller to allow the best accuracy possible for the controller, using either recommendations of motor manufacturers when available or tuning methods from the literature.		
Applied to the real systems: the custom parameters were based on the recommendation of the manufacturer.		
Step 2—Choose between INN and GNN methods. Estimate the ratio $\boldsymbol{J_m/(RMR)}$ (see a suggested method in Appendix A). Refer to the cases below.		
Case 1: the ratio is larger than 100: INN method will have a good accuracy in the trajectory tracking with a minimum number of parameters to train in the NN	**Case 2: the ratio is between 10 and 100, or the ratio is hard to estimate.** INN method should have a good accuracy, but it is advised to compare with the accuracy of GNN method if the accuracy of the system is crucial.	**Case 3: the ratio is smaller than 10:** Using GNN method is necessary to obtain good accuracy.
Applied to the real systems: the ratio is estimated to 194 (see Appendix A) for the 3-DOF system (case 1). Thus, we chose INN method for the 3-DOF system. For the 5-DOF system, the ratio is estimated to 31 (see Appendix A). Thus, we implemented both methods (case 2); as they both led to the same accuracy, we chose INN method.		
Step 3—Build the training dataset, following two recommendations: 3.1 The trajectories of the dataset should be randomly generated in the whole workspace of each motor: at least 100 trajectories for INN method or 100 trajectories for each DOF of the system for GNN method. 3.2 If a sufficient encoder resolution is available (the motor should be able to go through at least 20 steps of the encoder between two consecutive time steps), the step time of the PD controller should be equal to the step time of the NN architecture. Otherwise, the choice of the step time should be adapted to the motor encoder resolution.		
Applied to the real systems: To build the training dataset: 1—At least 100 trajectories per motor were randomly generated in the whole workspace of each motor. 2—The resolution of encoders (4096 steps/rev) and the maximum velocity of the motors (30 RPM) allow a maximum step time of 0.01s (20 steps of the encoder between each time step). Thus, while the PD controller was internal to our motors and with a step time of 0.001s, the step time of the training data was reduced to 0.01s.		

Step 4—Build and train the NN architecture.	
Case 1: INN method chosen at step 2, build INN, i.e., one NN for each joint of the system with only one neuron. If the training does not converge, it is possible to gradually increase the number of neurons.	**Case 2: GNN method chosen at step 2,** build one GNN, i.e., one NN for the whole robotic arm with at least as many neurons as the number of joints on each layer. If the training does not work, it is possible to gradually increase the number of neurons and the number of layers.
Then, train the NN architecture with all available data (at least 100 trajectories for INN method or 100 trajectories for each DOF of the system for GNN method)	
Applied to the real systems: the datasets of 100 trajectories per motor were used to train the real 3- and 5-DOF systems. The "MLPRegressor.fit()" function was used to train the NN. When the NN was trained with only 1 neuron, it did not converge. Thus, we chose to increase the number of neurons to 4 for the real system.	
Step 5—Add the trained NN in the control structure as a feedforward compensation to predict and correct the trajectory tracking errors.	
Applied to the real systems: The "MLPRegressor.predict()" function was used to generate the commands for new trajectories.	

White background refers to our experimental application of the guidelines while grey background refers to the general guidelines.

The main limitation of this study is that the control structure has been designed for fast trajectory tracking involving large movements and accelerations. For slow and micro-movements, the control structure is not optimized, and the accuracy could be improved by using an internal PID instead of a PD controller to correct static errors. However, a PID would be less accurate for fast trajectory tracking [27]. Future works could also explore more complex NN structures, for example, architectures specialized in time-series prediction as RNN or LSTM.

6.5 Conclusions

The objective of this paper was to design a minimal neural network (NN)-based model-free control structure for fast and accurate trajectory tracking of robotic arms. Particularly, the study compared the INN method by using one neural network for each joint of the robot arm and the GNN method by using a single global neural network for the entire system. The results were illustrated by two serial robotic arms with 3 and 5 degrees of freedom in a simulation, then, across several trajectory velocities ranging from 1 to 3 seconds.

The main results were as follows: 1. NN used as a feedforward compensation significantly reduced trajectory tracking errors (RMSE < 2°, see Table 2) without requiring any prior information about the dynamic model. 2. The study showed that the INN method can be used when joint dynamics are decoupled and requires three times less data than the GNN method to learn the dynamics. 3. Table 3 presents the guidelines for designing an optimal minimal NN-based control structure for accurate trajectory tracking in five main steps: 1. Tune the PD gains to optimize the accuracy of the PD controller. 2. Create an NN with an architecture that allows good accuracy with the fastest calculation. 3. Build a dataset representing the entire workspace of each motor. 4. Train the NN system by using all available data (at least 100 trajectories for the INN method or 100 trajectories per DOF of the system for the GNN method). 5. Use the trained NN in the control structure as a feedforward compensation to predict and correct the trajectory tracking errors.

Future perspectives will be to apply these guidelines to the development of 3D-printed low-cost robotic arms and extend them to other systems, such as automated visual inspection robots involving micro- and slow movements.

6.6 Statements and Declarations

Funding: Institute for data valorization (IVADO), Montreal, Canada.

Author contributions: Conceptualization: BT, MR Methodology: BT, MR. Investigation: BT, MR. Visualization: BT, MR. Funding acquisition: BT, MR. Project administration: BT, MR. Supervision: MR. Writing – original draft: BT. Writing – review & editing: BT, MR.

Competing interests: Authors declare that they have no competing interests.

6.7 Appendix

6.7.1 A1. Estimation of the $\boldsymbol{J_m/(RMR)}$Ratio

To quickly obtain an estimation of the magnitude of the impact of motor inertia $\boldsymbol{J_m}$ and the impact of the robot dynamics $\boldsymbol{RMR}$ on joint dynamics for a serial system, the estimation of the biggest value of the $\boldsymbol{RMR}$ matrix is proposed as follows:

$$\max(\boldsymbol{RMR}) = (\sum m_i d_i^2 + \sum I_i)\boldsymbol{R}^2$$

with m_i the mass of the i^{th} body, d_i being the maximum distance between the basis of the system and the center of mass of the i^{th} body, I_i being the biggest inertia of i^{th} body, and R being the gear reduction ratios of the motors.

Then, it is possible to compare with the value of the motor inertia.

6.7.2 A2. Case Example 1—Real 3-DOF Robotic Arm

For 3-DOF real system, the parameters were $m_1 = m_2 = m_3 = 0.25$ kg, $I_1 = I_2 = I_3 = 3.13\ 10^{-4}$ kgm^2, $d_1 = 0.05$ m, $d_2 = 0.15$ m, $d_3 = 0.25\ m$, and $R = 1/270$.

Thus, max($\boldsymbol{RMR}$) $= 3.13\ 10^{-7}$ kgm^2

Moreover, the inertia of the motor around the rotating axis is given by the manufacturer: $J_{mi} = 6.08\ 10^{-5}$ kgm^2. Finally, the ratio $\frac{J_{mi}}{\max(\boldsymbol{RMR})} = 194$.

6.7.3 A3. Case Example 2—Real 5-DOF Robotic Arm

For 5-DOF real system, the parameters were $m_1 = m_2 = m_3 = 0.3$ kg, $m_4 = 0.15$ kg, $m_5 = 0.1$ kg $I_1 = I_2 = I_3 = 3.75\ 10^{-4}$ kgm^2, $I_4 = I_5 = 6.75\ 10^{-5}$ kgm^2, $d_1 = 0$ m, $d_2 = 0.05$ m, $d_3 = 0.35$ m, $d_4 = d_5 = 0.65$ m, and $R = 1/270$.

Thus, max($\boldsymbol{RMR}$) $= 1.98\ 10^{-6}$ kgm^2. Finally, the ratio $\frac{J_{mi}}{\max(\boldsymbol{RMR})} = 31$.

6.8 References

[1] C. Taylor, “Robots could take over 20 million jobs by 2030, study claims.” Accessed: Jun. 25, 2024. [Online]. Available: https://www.cnbc.com/2019/06/26/robots-could-take-over-20-million-jobs-by-2030-study-claims.html

[2] M. Zhihong and M. Palaniswami, “Robust tracking control for rigid robotic manipulators,” *IEEE Transactions on Automatic Control*, vol. 39, no. 1, pp. 154–159, 1994.

[3] O. Koç, G. Maeda, and J. Peters, “Optimizing the execution of dynamic robot movements with learning control,” *IEEE Transactions on Robotics*, vol. 35, no. 4, pp. 909–924, 2019.

[4] S. Chen and J. T. Wen, “Neural-learning trajectory tracking control of flexible-joint robot manipulators with unknown dynamics,” in *International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 128–135. Accessed: Jun. 25, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8968608/

[5] S. Chen and J. T. Wen, “Industrial robot trajectory tracking control using multi-layer neural networks trained by iterative learning control,” *Robotics*, vol. 10, no. 1, p. 50, 2021.

[6] M. Spong, K. Khorasani, and P. Kokotovic, “An integral manifold approach to the feedback control of flexible joint robots,” *IEEE Journal on Robotics and Automation*, vol. 3, no. 4, pp. 291–300, 1987.

[7] S. Arimoto, “Learning control theory for robotic motion,” *Adaptive Control & Signal*, vol. 4, no. 6, pp. 543–564, Nov. 1990, doi: 10.1002/acs.4480040610.

[8] M. Norrlof, “An adaptive iterative learning control algorithm with experiments on an industrial robot,” *IEEE Transactions on robotics and automation*, vol. 18, no. 2, pp. 245–251, 2002.

[9] S. Chen and J. T. Wen, “Adaptive neural trajectory tracking control for flexible-joint robots with online learning,” in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 2358–2364. Accessed: Jun. 25, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9197051/

[10] W. He, Z. Yan, Y. Sun, Y. Ou, and C. Sun, “Neural-learning-based control for a constrained robotic manipulator with flexible joints,” *IEEE transactions on neural networks and learning systems*, vol. 29, no. 12, pp. 5993–6003, 2018.

[11] B. Toussaint and M. Raison, “High-speed trajectory tracking on robotic arm by learning the dynamic response of the pid controller with a neural network,” 2021, Accessed: Jun. 25, 2024. [Online]. Available: https://publications.polymtl.ca/50604/

[12] M. Wang, H. Ye, and Z. Chen, “Neural Learning Control of Flexible Joint Manipulator with Predefined Tracking Performance and Application to Baxter Robot,” *Complexity*, pp. 1–14, 2017, doi: 10.1155/2017/7683785.

[13] W. Lou and X. Guo, “Adaptive Trajectory Tracking Control using Reinforcement Learning for Quadrotor,” *International Journal of Advanced Robotic Systems*, vol. 13, no. 1, p. 38, Jan. 2016, doi: 10.5772/62128.

[14] Y. Ouyang, L. Dong, Y. Wei, and C. Sun, “Neural network based tracking control for an elastic joint robot with input constraint via actor-critic design,” *Neurocomputing*, vol. 409, pp. 286–295, 2020.

[15] Y. Jiang, Y. Wang, Z. Miao, J. Na, Z. Zhao, and C. Yang, “Composite-learning-based adaptive neural control for dual-arm robots with relative motion,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 3, pp. 1010–1021, 2020.

[16] L. Kong, W. He, C. Yang, and C. Sun, “Robust neurooptimal control for a robot via adaptive dynamic programming,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 6, pp. 2584–2594, 2020.

[17] W. He, Y. Dong, and C. Sun, “Adaptive neural impedance control of a robotic manipulator with input saturation,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 3, pp. 334–344, 2015.

[18] J. H. Pérez-Cruz, I. Chairez, J. Rubio, and J. Pacheco, “Identification and control of class of non-linear systems with non-symmetric deadzone using recurrent neural networks,” *IET Control Theory & Applications*, vol. 8, no. 3, pp. 183–192, Feb. 2014, doi: 10.1049/iet-cta.2013.0248.

[19] S. J. Yoo, J. B. Park, and Y. H. Choi, “Adaptive output feedback control of flexible-joint robots using neural networks: dynamic surface design approach,” *IEEE Transactions on Neural Networks*, vol. 19, no. 10, pp. 1712–1726, 2008.

[20] H. A. Talebi, R. V. Patel, and K. Khorasani, “Inverse dynamics control of flexible-link manipulators using neural networks,” in *Proceedings. International Conference on Robotics and Automation (Cat. No. 98CH36146)*, IEEE, 1998, pp. 806–811. Accessed: Jun. 25, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/677084/

[21] R. Calandra, S. Ivaldi, M. P. Deisenroth, E. Rueckert, and J. Peters, “Learning inverse dynamics models with contacts,” in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 3186–3191. Accessed: Jun. 25, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7139638/

[22] E. Rueckert, M. Nakatenus, S. Tosatto, and J. Peters, “Learning inverse dynamics models in o (n) time with lstm networks,” in *17th International Conference on Humanoid Robotics (Humanoids)*, IEEE, 2017, pp. 811–816. Accessed: Jun. 25, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8246965/

[23] C. Yang, X. Wang, L. Cheng, and H. Ma, “Neural-learning-based telerobot control with guaranteed performance,” *IEEE transactions on cybernetics*, vol. 47, no. 10, pp. 3148–3159, 2016.

[24] T. Sun, H. Pei, Y. Pan, H. Zhou, and C. Zhang, “Neural network-based sliding mode adaptive control for robot manipulators,” *Neurocomputing*, vol. 74, no. 14–15, pp. 2377–2384, 2011.

[25] J. Trierweiler, “A systematic approach to control structure design,” 1997.

[26] Q. Li, J. Qian, Z. Zhu, X. Bao, M. K. Helwa, and A. P. Schoellig, “Deep neural networks for improved, impromptu trajectory tracking of quadrotors,” in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 5183–5189. Accessed: Jun. 25, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7989607/

[27] S. Kawamura, F. Miyazaki, and S. Arimoto, “Is a local linear PD feedback control law effective for trajectory tracking of robot motion?,” in *Proceedings. International Conference on Robotics and Automation*, IEEE, 1988, pp. 1335–1340. Accessed: Jun. 25, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/12253/

[28] N. Docquier, A. Poncelet, and P. Fisette, “ROBOTRAN: a powerful symbolic gnerator of multibody models,” *Mechanical Sciences*, vol. 4, no. 1, pp. 199–219, 2013.

[29] P. Rocco, “Stability of PID control for industrial robot arms,” *IEEE transactions on robotics and automation*, vol. 12, no. 4, pp. 606–614, 1996.

[30] “scikit-learn: machine learning in Python — scikit-learn 1.5.0 documentation.” Accessed: Jun. 25, 2024. [Online]. Available: https://scikit-learn.org/stable/

[31] B. Toussaint, “LiBRTy-Lab/minimal_neural_controller: Public code of paper : Design of a minimal model-free controller for fast robotic arms trajectory.” Accessed: Jul. 16, 2024. [Online]. Available: https://github.com/LiBRTy-Lab/minimal_neural_controller

CHAPITRE 7 ARTICLE 4 : REAL-TIME ALGORITHM FOR TABLE TENNIS WITH A DESKTOP ROBOTIC ARM

Submitted to *Robotics and Autonomous Systems*, on April 18, 2025.

Abstract: Table tennis with collaborative robots has been a challenge in robotics for decades, due to its unique challenges, especially high-speed movements and real-time ball trajectory predictions for responsive and accurate gameplay. Over the years, several table tennis robots have been developed, showing progressively enhanced abilities for returning balls, hitting specific targets, rallying with collaborative human users, and playing amateur-level games.

However, these robotic systems remain costly for individuals, often relying on industrial components, or specialized designs. Emerging AI-integrated personal desktop robotic arms could help bridge the performance gap between affordable personal robotic systems and traditional industrial robots, particularly in terms of dexterity, speed, and precision. Despite this potential, desktop robotic arms have not yet been used for table tennis. However, existing table tennis algorithms require specific adaptations to accommodate the constraints of desktop robots.

This paper aims to develop a dedicated algorithm for a collaborative table tennis system using a desktop robotic arm to demonstrate the achievable performance of AI-integrated desktop robots. The proposed system utilizes a 5-degree-of-freedom (DOF) serial robot, integrating advanced algorithms and machine learning models to improve performance.

This system enables short collaborative rallies, returning 71.3% of balls overall, improving to 81.4% after fine-tuning system parameters – approaching the best one from the literature (88%) using a 7-DOF industrial robotic arm. This underscores the potential of affordable, AI-integrated desktop robotic arms for high-speed human-robot collaboration. Future works will focus on adapting the algorithm for specialized desktop hardware, expanding desktop robots to other applications, and further enhancing their performance.

Keywords: machine learning, neural networks, trajectory prediction, 3D-printing, motor control, inverse kinematics

7.1 Introduction

With increasing computational power, robots have made significant progress, enabling them to perform daily tasks, such as cooking [1] or cleaning [2], or even executing complex maneuvers like backflips [3]. While the capabilities of learned robot policies have significantly increased, achieving human—level performance in terms of accuracy, speed, and generalizations remains a major challenge across many domains [4]. One such domain is table tennis, which combines challenges in real—time sensing, intelligent decision-making, high-speed motion control, and robotic design. Thus, table tennis has been a focus in robotics research since the 1980s [5].

Since the development of the first ping-pong-playing robot in 1988 [6], numerous robotic systems have been designed to play table tennis, progressively improving their skills [7], [8], [9], [10], [11]. While some systems address the complete game, many studies have focused on specific aspects, such as trajectory prediction [12], racket pose detection [13], spin detection [14], [15] and identifying human strategies [16], [17], [18]. Other research focused on motion generation and actions, such as returning the ball [19], [20], [21], [22], [23], targeting to a specific area [24], [25], rallying [26], [27], [28] and even smashing [11], [29].

The performance of robotic systems for table tennis is evaluated through several metrics. First, the most common is the ball return rate (percentage of balls successfully returned to the user) [10], [19], [28]. Depending on the capabilities of the system, this metric can be combined with the hit rate [25], or an RMSE between the target position and the rebound position of the outgoing ball [30]. For competitive table tennis systems, performance may also be measured by the win rate against human players of varying skill levels [4], [31]. Finally, all these metrics can be obtained under different conditions of play: against a competitive human player [4], a collaborative human user [10], [28], [30], with another robot [28], or even with a ball launcher [25].

Table 1 summarizes the notable existing robotic table tennis systems.

Tableau 7.1 : Structure of existing robotic table tennis system, with their performance

Model (Company, City, Country)	**Paper date**	**Architecture summary**	**Global performance**
IRB 1100 DeepMind system (Google, Mountain View, USA)	2024	Industrial serial 6-DOF arm mounted on top of 2 Festo linear gantries	First competitive player, “amateur human-level performance in competitive table tennis,” wins 45% of the matches vs amateur players [4]

Forpheus (Omron, Kyoto, Japan)	2019	Industrial 5-DOF parallel robot solely designed for table tennis. Delta robot suspended over the table + 2 DOF for the paddle	Extended rallying with humans, feedback to human player regarding their performance. RMSE 22.5cm [30]
Agilus (KUKA, Tübingen, Germany)	2018	Industrial serial 6-DOF arm	87% of the balls back to the table, over 315 strokes [10]
WAM (Barrett, Tübingen, Germany)	2010-2013	Industrial serial 7-DOF arm	Returns 74.4 % of the balls. After training, return 88 % of the balls. [16]
PA10 (Mitsubishi, Nagoya, Japan)	2012	Industrial serial 7-DOF arm	70% of ball hit, ball launched using a ball throwing machine. [25]
Humanoïd (Zhejiang University, Hangzhou, China)	2012	30-DOF with 7-DOF per arm	145 hits with a collaborative user. [28]

Currently, the two setups that have distinguished themselves the most are the Forpheus (Omron, Kyoto, Japan) [30], and the IRB 1100 DeepMind system (Google, Mountain View, USA) [4], as detailed in the paragraphs below.

Forpheus is a robot that can engage in ping-pong rallies with human players [30]. This system features a racket mounted at the end of a suspended Delta robot, located above the table. Its architectures allow for excellent reactivity, high accelerations and precision. Forpheus uses a model-based approach, identifying the optimal configuration for returning the ball to a target location through dynamic models that consider ball rebound and aerodynamics. Forpheus demonstrated its ability to maintain rallies across various stroke styles with skilled human players, while also providing performance feedback to the human players.

In contrast, IRB 1100 DeepMind is an adaptive robotic system capable of playing engaging and skilled games of table tennis against human opponents. This system is the first to play full competitive matches against players with a wide range of skill levels. It consists of a 6-DOF IRB 1100 industrial robot (ABB, Zurich, Switzerland), mounted on top of two Festo linear gantries, allowing the robot base to move in a 2D plane [31]. Contrarily to Forpheus, this system learns the control policies and perception through a combination of simulations and real-world data [24], [26], eliminating the need for explicit dynamic models in the final system. The results showed that this learned robotic agent *"achieves amateur human-level performance in competitive table tennis"* [4].

Despite these advances, a common issue with these robotic systems is their lack of affordability for individual users and so hindering the democratization of such systems in society. In fact, all table tennis robotic systems rely either on industrial robots and components, or complex systems solely designed for table tennis, also increasing costs. For example, the ABB arm used in [4] costs around 25,000 US $, while the 6-DOF KUKA Agilus exceeds 50,000 US $ [10].

Additionally, despite Anderson's assertion that only 5-DOF are required for a robot paddle to execute table tennis tasks [6], most table tennis robotic systems feature a redundant number of DOF, except for Forpheus. In [32], [33], two similar prototypes with 5-DOF were specifically designed for table tennis. However, these systems are complex, incorporating two rackets mounted on a 2D-Cartesian vertical structure to cover the entire playing area while maintaining sufficient reactivity. Each racket has an additional 3-DOF to return the ball, requiring a total of 8 motors for the entire system. The DOF redundancy also results in costly setups.

Moreover, some systems must operate in controlled environments to detect the ball or track the player's racket movements [30]. To date, no work has explored the use of a low-cost, minimal robotic arm for table tennis with a collaborative user.

Desktop robotic arms offer a promising solution for reducing the cost of robotic systems. For example, a 3-printed 5-DOF serial desktop robotic arm was developed by our laboratory for under than 3,500 US $ and was used for high-speed trajectory tracking [34]. However, using a desktop robotic arm for table tennis requires optimizing and adapting existing algorithms to accommodate the limitations of these robotic arms with reduced capabilities. In fact, algorithms designed for redundant industrial robots often rely on simplifications. For example, the hitting point between the ball and the racket is usually determined arbitrarily using a virtual hitting plane (VHP) [10], [30], which is unsuitable for desktop robots with limited workspace. These robots also usually follow rectilinear trajectories before hitting the ball to increase robustness [10], [30], a strategy that is not adapted for serial robots with no redundant DOF.

Consequently, if desktop robotic devices have been used for academic purposes for a variety of investigation and development studies [35], [36], [37], [38], current low-cost desktop robots are mostly limited to tasks requiring fewer dynamic movements and lower accelerations, such as pointing or writing [39]. To the best of our knowledge, no desktop robotic arm has yet been used to play table tennis.

Thus, the objective of this study is to develop a dedicated algorithm for a collaborative table tennis system using a desktop robotic arm to demonstrate the achievable performances of AI-integrated desktop robotic arms.

The approach uses a 5-degrees-of-freedom 3D-printed serial desktop robotic arm, integrating state-of-the-art methods adapted to the constraints of this robotic platform.

7.2 Description of the solution

This section is organized as follows: first, the hardware architecture of the robotic system is explained (section 2.1). Secondly, the general structure of the algorithm is described (section 2.2). Thirdly, the modules of the code are detailed, including the ball detection process, the ball trajectory prediction (section 2.3), the robot "planner" process (section 2.4) and the algorithm used for motor control (section 2.5). Finally, the evaluation strategy is explained (section 2.6).

7.2.1 System architecture

The vision system consisted of three Optitrack Flex3 infrared cameras operating at 100 Hz, located around the table. Image processing was managed by Motive software, provided by Optitrack (Corvallis, USA) [40]. To ensure visibility to the vision system, the ball was covered with infrared tape.

The desktop robotic arm in the system was a 5-DOF, 3D-printed robot, mounted on top of a metal structure behind the table. Dynamixel motors from the XM series powered the joints. XM-540 motors were used on the first three joints (q1, q2, q3), while XM-430 motors were installed on the last two joints (q4 and q5). These motors provided maximum speeds of 30 and 77 revolutions per minute (RPM), with stall torques of 10.6 Nm and 3Nm, respectively [41].

To support the weight of the arm, the second joint motor was doubled, with two XM-540 operating in master-slave mode. A 3D-printed paddle was rigidly fixed at the extremity of the arm. Figure 1 presents the topology of the arm.

The entire system was controlled by a computer with an Intel Core i7-5500U CPU, and 16GB of RAM. The proposed algorithm was implemented using Python.

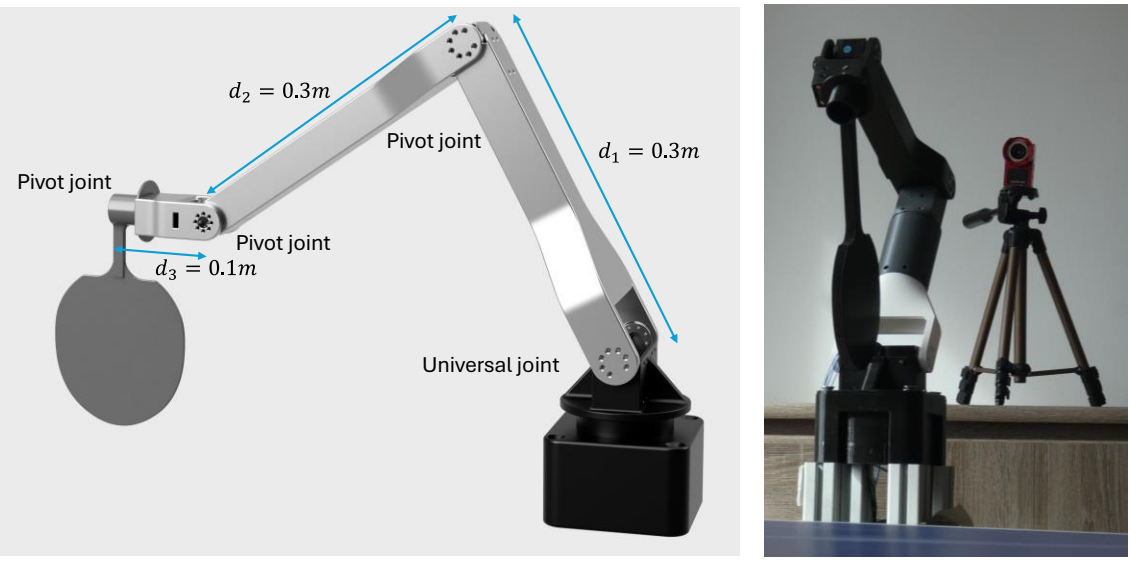


Figure 7.1 : A. Topology of the table tennis robotic arm. B. Real 3D-printed robot during play

7.2.2 Algorithm overview

The default state of the algorithm involved the robot waiting for an incoming ball. During this phase, the vision system continuously provided the algorithm with successive ball positions. A ball was considered “incoming” when 10 consecutive ball positions indicated that it was approaching the robot. Once this condition was met, the state changed, and the remainder of the algorithm was activated.

The algorithm was organized into three main blocks operating independently. These are summarized in Figure 2.

First, a ball’s trajectory prediction module estimated the trajectory of the incoming ball from previous camera observations. Secondly, the predicted trajectory was fed into the robot planner. The planner selected the optimal hitting parameters while ensuring they fell within hardware constraints, such as hit position, racket speed, and joint position, and generated the robot trajectory to perform the hit. The planner consisted of four modules: 1. a module which identified potential hitting points along the predicted ball trajectory, 2. a module which computes possible racket orientations and speeds at the point of contact, 3. the inverse kinematic module, which calculated the corresponding joint positions and speeds for each configuration and selected the optimal one,

and 4. the trajectory generation module, which generated the joint trajectories required to execute the hit.

Thirdly, the robot control structure ensured accurate trajectory tracking the selected hitting configuration identified by the planner.

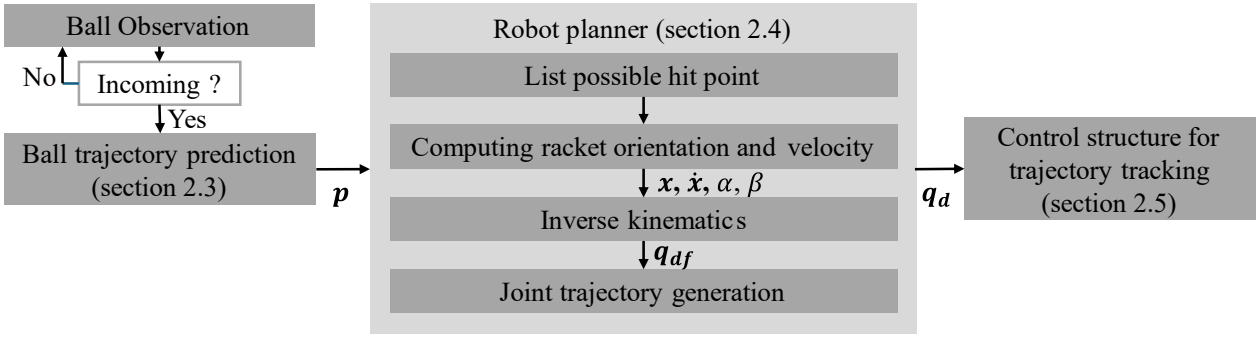


Figure 7.2 : Structure of the algorithm of the table tennis robot

7.2.3 Ball trajectory prediction

An essential task for the robot is to accurately predict the future trajectory of an incoming ball based on previous visual observations [42]. These visual measurements are often noisy, with outliers and sparse data [43]. Moreover, the computation time — or latency — of the predictions, plays a major role, as it must be significantly faster than the ball itself [44].

Methods for predicting the trajectory of a ping-pong ball have been widely studied and are summarized in [45]. They can be grouped into two categories: model-based and data-driven methods. A comparative analysis of data-driven methods was conducted in [45], highlighting their effectiveness. Additionally [45], presented a hybrid approach that enhances a model-based method using machine learning optimization.

This study adopts the trajectory prediction method proposed in [45], which utilizes a model-based approach without spin estimation, incorporating optimized dynamic parameters. This method was chosen for its compatibility with our hardware architecture. First, it can be easily adapted to account for non-standard dynamic parameters, such as those of the ball used in our system, which includes reflective tape. Secondly, after optimization, the trajectory prediction has an average computation time of only 1.8ms, enabling real-time operation while continuously updating predictions as the ball moves toward the robot. For comparison, additional methods from [45] were also implemented, including data-driven approaches such as a gated-recurrent-unit trajectory

autoencoder (GRU-TAE). Further details on the optimization method implementation can be found in [45].

7.2.4 Robot planner

The robot planner is responsible for selecting the optimal hitting conditions based on the predicted ball trajectory while accounting for the specific limitations of the desktop robotic arm. It must be adapted from existing algorithms to suit this hardware.

Since the standard method of determining the hitting point using a virtual hitting plane (VHP) [10], [30] is too restrictive due to the robot's limited workspace, the first step of the developed algorithm is to identify and select feasible hitting points along the predicted ball trajectory. To achieve this, all positions along the computed ball's trajectory after the rebound are computed. Positions that are beyond the robot's reach are discarded, leaving only those within a feasible striking range.

Next, for each identified ball position, the planner calculates the speed and the orientation for the paddle at the selected hitting points to return the ball to the user in a desired position p_2 at a specific time t_2. This is done using standard equations from the literature [30]:

$$\begin{bmatrix} \boldsymbol{R^T} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{R^T} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{p}}_{\boldsymbol{1}} - \dot{\boldsymbol{x}} \\ \boldsymbol{\omega_1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A_{vv}} & \boldsymbol{A_{v\omega}} \\ \boldsymbol{A_{\omega v}} & \boldsymbol{A_{\omega\omega}} \end{bmatrix} \begin{bmatrix} \boldsymbol{R^T} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{R^T} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{p}}_{\boldsymbol{0}} - \dot{\boldsymbol{x}} \\ \boldsymbol{\omega_0} \end{bmatrix} \quad (\mathbf{1})$$

Here, $\dot{p}_0$, ω_0, $\dot{p}_1$, ω_1 represent the velocity and spin of the incoming and the outgoing ball respectively. $\dot{p}_1$, ω_1 are determined using p_2 and equation (1) describing the flight phase of the ball. $\dot{x}$ is the racket velocity at the instant of the impact, R stands for the rotation matrix from the global coordinate system to the racket coordinate system, and is given by:

$$\boldsymbol{R} = \begin{bmatrix} \cos\alpha\cos\beta & -\sin\alpha & \cos\alpha\sin\beta \\ \sin\alpha\cos\beta & \cos\alpha & \sin\alpha\sin\beta \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \quad (\mathbf{2})$$

Finally, $\boldsymbol{A_{vv}}$, $\boldsymbol{A_{v\omega}}$, $\boldsymbol{A_{\omega v}}$, $\boldsymbol{A_{\omega\omega}}$ represent the transformation matrices between the incoming and outgoing ball and are defined as:

$$\boldsymbol{A_{vv}} = \begin{bmatrix} -k_e & 0 & 0 \\ 0 & 1-k_v & 0 \\ 0 & 0 & 1-k_v \end{bmatrix}, \boldsymbol{A_{v\omega}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & k_v r \\ 0 & -k_v r & 0 \end{bmatrix}$$

$$\boldsymbol{A_{\omega v}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -k_\omega r \\ 0 & k_\omega r & 0 \end{bmatrix}, \boldsymbol{A_{\omega\omega}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1-k_\omega r^2 & 0 \\ 0 & 0 & 1-k_\omega r^2 \end{bmatrix} \quad (\mathbf{8})$$

With k_e the restitution coefficient between the ball and the racket, k_v and k_ω the friction coefficients due to the relative velocity and spin between the ball and the racket. Using this set of equations, the racket velocity $\dot{\boldsymbol{x}}$, and its orientation angles α and β can be determined [30]. However, without considering the spin of the ball, there are only three equations for five variables, making it impossible to obtain a unique solution. Therefore in [30], two constraints conditions $\dot{\boldsymbol{x}}_{\boldsymbol{z}} = 0$ and $\beta = 0$ are imposed: $\dot{\boldsymbol{x}}_{\boldsymbol{z}}$ is the vertical racket velocity swing, which is a motion relatively insignificant in the attempt to improve the accuracy of the position of the return shot; and β, is the elevation angle of the racket. Figure 4 illustrates these parameters.

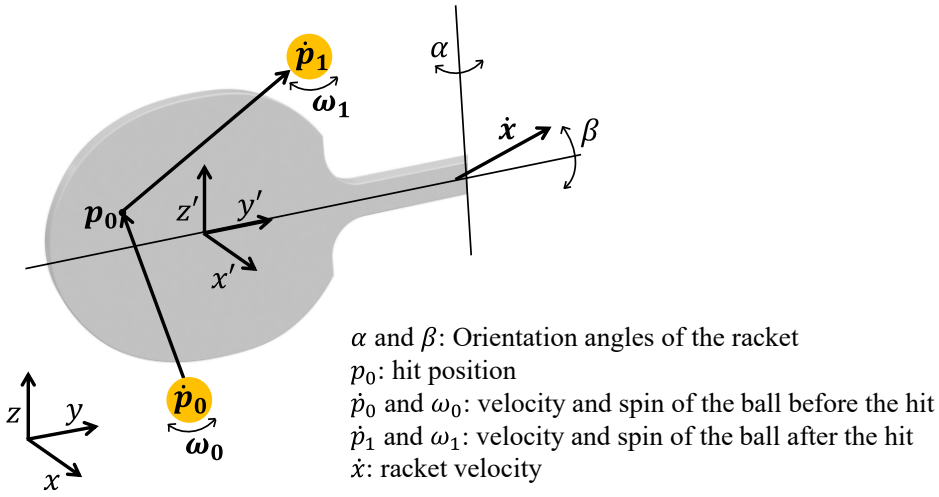


Figure 7.3 : Conceptual image of the racket motion planning during a hit.

For our 5-DOF system, imposing such constraints would again be too restrictive. Instead, we iterated over α and β to explore a wider range of possible configurations and identify the optimal one. For each selected pair of α and β, a symbolic solution can be computed. Given that $k_e = 0.6546$ and $k_v = 0.615$, we used MATLAB Symbolic Math Toolbox [46] to derive the following solution:

$$\dot{x}_x = [v_{1x} - v_{0x} + k_e(v_{1x} - v_{0x}) + k_v v_{0x} + (v_{0x} - v_{1x})\cos^2\alpha\cos^2\beta + k_e k_v v_{0x} + (1 + k_e - k_v)(v_{1z} - v_{0z})\cos\alpha\cos\beta\sin\beta + (k_e(v_{0x} - v_{1x}) + k_v(v_{1x} - v_{0x}))\cos^2\alpha\cos^2\beta + (1 + k_e - k_v)(v_{0y} - v_{1y})\cos\alpha\cos^2\beta\sin\alpha]/(k_v(k_e + 1))$$

$$\dot{x}_y = [v_{1y} - v_{0y} + k_e(v_{1y} - v_{0y}) + (k_v + k_e k_v)v_{0y} + (1 + k_e - k_v)(v_{0y} - v_{1y})\cos^2\beta + (1 + k_e - k_v)(v_{1y} - v_{0y})\ \cos^2\alpha\cos^2\beta + (1 + k_e - k_v)(v_{1z} - v_{0z})\cos\beta\sin\alpha\sin\beta + \ (1 + k_e - k_v)(v_{0x} - v_{1x})\cos\alpha\cos^2\beta\sin\alpha]/(k_v(k_e + 1))$$

$$\dot{x}_z = [k_v v_{1z} + (1 + k_e - k_v)(v1_z - v_{0z})\cos^2\beta + k_e k_v v_{0z} + (1 + k_e - k_v)(v_{1x} - v_{0x})\cos\alpha\cos\beta\sin\beta + (1 + k_e - k_v)\big(v_{1y} - v_{0y}\big)\cos\beta\sin\alpha\sin\beta]/(k_v(k_e + 1))$$

To minimize computation, for all possible hitting points, α and β, matrix calculations were implemented. Additionally, to further reduce computational complexity, possible racket hitting speeds were filtered, retaining only the slowest speed at each potential hitting point.

Thirdly, after determining the hitting conditions, the planner solves inverse kinematics equations for all hitting points to identify feasible configurations that satisfy hardware constraints. Once again, the inverse kinematics algorithm was optimized to compute all configuration in a single step, using matrix calculations. The best configuration—defined as the one requiring the least effort from the robotic arm—is then selected. To achieve this, the following loss function *L* is computed for all possible configurations.

$$L(q, V) = 10^6 \sum_{i=1}^{n} \max(0, (|\dot{q}_i| - \dot{q}_{i\,lim})^2) + 10^3 \left\|\boldsymbol{\omega_{raq}}\right\| + \sum_{i=1}^{n} (q_i - q_{i\,home})^2) \qquad (\mathbf{9})$$

Where $\dot{q}_i$ is the speed of the i-th motor, and $\dot{q}_{i\,lim}$ its maximum speed. $q_{i\,home}$stands for the home position of the i-th motor. Finally, $\boldsymbol{\omega_{raq}}$ is the rotation speed of the racket.

This loss function minimizes the rotational speed of the racket at the instant of the hit while ensuring the joint velocities remain within motor limits. If no configuration meets the speed limit, the function selects the one that minimizes the degree to which the limit is exceeded. If multiple configurations without racket rotation exist, the function selects the one closest to the home position.

Finally, based on the selected hitting configuration, the planner generates the robot arm trajectory. As in [10], [30], the robotic arm first moves to an intermediate position near the final hitting point to minimize positional errors of the robot and improve robustness. However, the algorithm for

trajectory generation must be adapted, as following a rectilinear trajectory—as in [10], [30]—would be too demanding for a 5-DOF desktop robotic arm. Instead, joint trajectories are generated using quintic polynomials, starting from the home position, using the desired speed and position for each motor.

Quintic polynomials enable precise control over the boundary conditions of the trajectory, i.e., the initial and final positions, velocities, and accelerations, while minimizing jerk, and reducing motor efforts. The polynomial is defined as follows:

$$q_i(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5, \quad (\mathbf{10})$$

with

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & t_d & t_d^2 & t_d^3 & t_d^4 & t_d^5 \\ 0 & 1 & 2t_d & 3t_d^2 & 4t_d^3 & 5t_d^4 \\ 0 & 0 & 2 & 6t_d & 12t_d^2 & 20t_d^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} q_{i0} \\ \dot{q}_{i0} \\ \ddot{q}_{i0} \\ q_{id} \\ \dot{q}_{id} \\ \ddot{q}_{id} \end{bmatrix} \quad (\mathbf{11})$$

where q_{i0} is the home position of the i^{th} motor, $\dot{q_{i0}} = 0$ and $\ddot{q_{i0}} = 0$ are the initial speed and acceleration of the motor, q_{id}, $\dot{q_{id}}$ are the desired position and speed while $\ddot{q_{id}} = 0$ is the desired acceleration.

The same equations were used to generate the second half of the trajectory between q_{id} and $q_{if} = q_{i0}$. All these trajectories were generated at a frequency of 100 Hz to ensure regular command transmission to the motors with our computer. In addition, the duration of the trajectories between q_0 and q_d was fixed at 0.5s to increase the accuracy of the learning process.

Since the computation time from ball trajectory prediction to robot trajectory generation has been minimized, the predictions can be updated several times between the initial detection and the hit. As the robot begins to move only 0.5s before the hit, the planner continuously updates the hitting point and conditions using the same process. While the robot is in motion, the cameras continue to track the ball's position to detect the rebound. Once the ball bounces, the planner updates the hitting conditions one final time without altering the time of the hit. This modification enables the system to account for the spin effect on the trajectory, particularly affecting the rebound.

7.2.5 Motor control

Once the desired trajectory has been generated for each joint, the control module ensures accurate trajectory tracking by the robotic arm. Industrial robots often include an internal control structure and are calibrated by the seller before the first use, the dynamic model of desktop robots cannot be estimated accurately.

Then the implemented control structure is shown in Figure 4, adapted from one of our previous works [34]. This model-free control architecture is specifically designed for high-speed trajectory tracking, combining the simplicity and stability of a proportional—derivative (PD) controller with neural networks effective to learn the dynamics of the systems. This approach has the advantage of not requiring any prior knowledge of the robot's dynamic model, making it particularly suitable for desktop robotic arms.

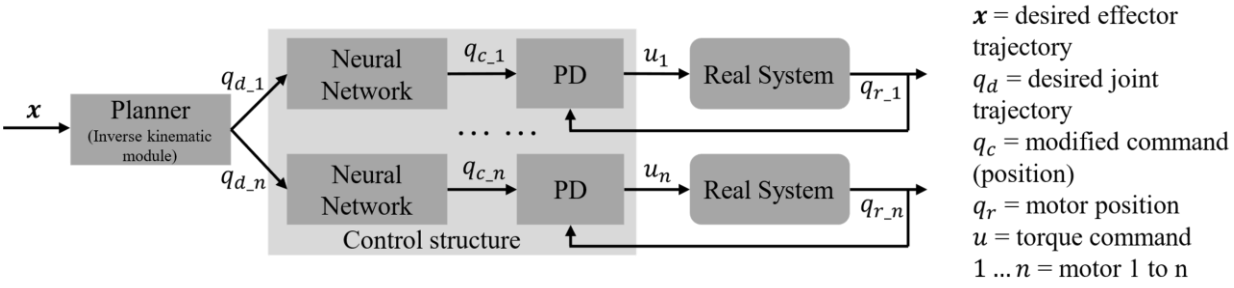


Figure 7.4 Model-free control structure of the robotic arm.

The architecture consists in learning the dynamic response of the PD controller using a NN, which is then applied in an outer loop as feedforward compensation to predict/correct the errors of this PD. This approach has been shown to improve trajectory tracking for robotic arms [34].

To train the NN, a dataset composed of 1000 trajectories was generated. Joint trajectories were executed from a home position using quintic polynomials, with randomly selected target speeds and positions within the workspace of each motor. Once these trajectories were executed, the dataset was built using all the available data. Each entry in the dataset consisted of five variables for each of the five motors: the motor position q and velocity $\dot{q}$ at two consecutive timestamps — $q(t)$, $q(t+\delta)$, $\dot{q}(t)$, $\dot{q}(t+\delta)$ – and the command $u(t)$ sent between these two timestamps.

The NN was then trained to learn the system dynamics by learning the PD controller response. By exposing the NN to a large number of trajectory data, it learned which command must be sent to

each motor to transition from a state — $q(t)$, $\dot{q}(t)$ – to the next state – $q(t+\delta)$, $\dot{q}(t+\delta)$ – thereby implicitly learning the dynamic model of the robotic arm. The multi-layer perceptron regressor was trained for a maximum of 500 iterations, using the "MLPRegressor.fit()" function from the Python library "sklearn" [47]. This function used as a loss function L, defined as:

$$L = \sum(\boldsymbol{u}(t) - \boldsymbol{q}(t+\delta)^2 \qquad (\mathbf{12})$$

The loss function was minimized using the Adam solver. The implementation and guidelines for the optimization of hyperparameters to maximize the accuracy of this control structure can be found in [34].

7.2.6 Methodology

To evaluate the performance of the adapted algorithm, the real-world performance of the robotic desktop system was evaluated by playing with a collaborative user. Standard metrics were used (see Table 1): return rate [10], [19], [28], hit rate [25], and miss rate. A return is considered successful if the ball is returned to the user's side of the table, following standard table tennis rules, defined as a bounce on the user's half of the table [48]. A "hit" is considered successful if the paddle contacts the ball launched by the user while a "miss" is when the paddle fails to contact the ball.

The robot was evaluated under two conditions: rallying, where it continuously returned balls to the user in a sequence, and non-rallying, where it returned one ball at a time to the user.

In addition, the performances were measured for several configurations of return parameters, varying the position of the base of the robot and positions aimed for the rebound of the ball. The results show the global performance across all tested configurations, and the performance obtained

with the best set of return parameters: $\boldsymbol{p_{base}} = [-0.68; 0; 0.1]$, and $\boldsymbol{p_2} = [2.2; 0; 0]$, $t_2 = 0.8$. Figure 6 illustrates these parameters.

In a second step, each module of the algorithm was evaluated separately to validate its

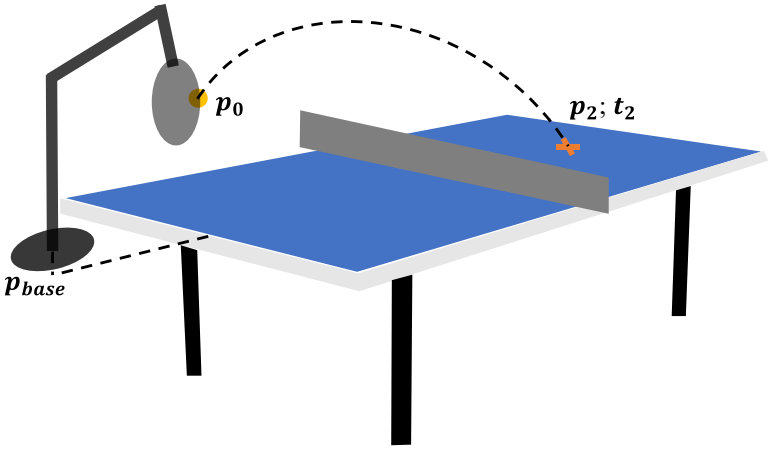


Figure 7.5 : Return parameters of the robotic system.

implementation and the adaptations. Since the algorithm operates in real-time, computation times for each section were carefully analyzed. Additionally, each module was assessed using specific performance metrics summarized in Table 2. This evaluation also helped identify potential performance limitations within the algorithm.

Tableau 7.2 : Metrics used for evaluation of modules of the code

Method	Metrics
Ball trajectory prediction	RMSE of the prediction (mm), computation time (ms)
Robot planner	Number of tested solutions and computation time of all modules
Robot control	RMSE during trajectory tracking (deg)

7.3 Results

7.3.1 Global performances with a collaborative user

Table 3 shows the performance of the table tennis robotic arm during rallying with a collaborative user over 526 throws. Additionally, the longest rally performed with the robot was 32 consecutive hits.

Tableau 7.3 : Summary of performances of our table tennis robotic arm during rallying

Performance metrics	**Global, overall tested configurations**	**With optimized return parameters: p_{base}, p_2 and t_2**
Return rate (number of occurrences)	71.3% (375)	81.4% (57)
Hit rate (number of occurrences)	94.7% (498) •Net ball: 12.3% (65) •Long return: 4.6% (24) •Side ball: 3.4% (18) Other: 3.0% (16)	95.7% (67)
Miss rate (number of occurrences)	5.3% (28)	4.3% (3)

Table 4 shows the performance of the table tennis robotic arm when returning balls to a collaborative user, in non-rallying conditions, one at a time. In addition, the robot was able to return up to 21 consecutive throws from the user without rallying.

Tableau 7.4 : Performances of our table tennis robotic arm in non-rallying conditions, one shot at a time

Performance metrics	**Global, overall tested configurations**	**With optimized return parameters: p_{base}, p_2 and t_2**
Return rate (number of occurrences)	80.2% (179)	89.4% (51)
Hit rate (number of occurrences)	97.7% (218)	100% (57)
Miss rate (number of occurrences)	2.3% (5)	0.0% (0)

7.3.2 Module Performances

7.3.2.1 Ball trajectory prediction

Table 5 presents the performances of our prediction method, along with other tested methods for comparison. Our system, which utilizes an optimized dynamic model without spin estimation, achieves an RMSE of 65 mm on real-world trajectory predictions over 0.9 seconds—the typical duration between ball observation and the robot's hit—while maintaining a computation time of just 1.8 ms.

In comparison, the reference method from [45] which combines a model-based approach with spin estimation with TAE and GRU-TAE achieves an RMSE of 49 mm representing a 25% improvement, but with a computation time of 53 ms — 29.4 times longer. When the same method neglects spin estimation, the RMSE increases to 53 mm, with a computation time of 12 ms. Removing the GRU-TAE results in an RMSE of 54 mm, with a computation time of 3.8 ms. GRU-TAE and TAE achieve RMSEs of 64 and 65 mm, with computation times of 8.2 and 2 ms, respectively. Meanwhile, model-based methods incorporating spin estimation, produce RMSEs of 58 mm, with computation times of 43 ms.

Tableau 7.5 : Summary of the performances of the ball trajectory prediction module

Method	**Global RMSE of the trajectory prediction, on first 0.9 s (mm)**	**Computation time (ms)**
Proposed method (TAE + no spin)	**54**	**3.8**
no spin + GRU-TAE + TAE	53 (-2%)	12 (x 3.2)
Hybrid^3 (Spin + GRU-TAE + TAE)	49 (-9%)	53 (x 13.9)
GRU-TAE	64 (+19%)	8.2 (x 2.2)
TAE	65 (+20%)	2 (x 0.53)
Spin	58 (+7%)	43 (x 11.3)
No spin	65 (+20%)	1.8 (x 0.47)

7.3.2.2 Robot planner

The robot planner can handle up to 2500 different hitting configurations and compute the required racket settings at the impact to ensure a successful return. Table 6 summarizes the computation times for the planner modules. The first module identifies around 25 potential hitting points in under 1 ms. For each possible hitting point, 100 racket orientations, defined by the set of values $(\alpha; \beta)$ are computed, also in under 1 ms.

Then it takes an average of 6 ms to compute the required speed for the racket across all configurations. Finally, the inverse kinematic module finds the optimal joint configuration from the 25 tested configurations in approximately 1 ms.

In total, the entire planner module requires an average of 7 ms to compute the desired robot trajectory based on the predicted ball trajectory.

Tableau 7.6 : Summary of the performances of the planner module

Algorithm	**Number of tested solutions**	**Average computation time (ms)**
Possible hit points	Around 25	< 1
Possible racket orientations	100 x number of possible hitting point	< 1
Racket speed	Around 2500	6
Inverse kinematic	Around 25	1
Trajectory generation	1	< 1
Total		**7**

7.3.2.3 Robot control

Experiments using our control system showed an RMSE of 0.76 degrees for slow joint trajectory tracking on our 5-DOF robotic arm, and 0.88 degrees for the last three joints during full-speed trajectory tracking [34]. In the worst-case scenario, an error of 0.88 degrees per joint would result in a position error of 3.1 centimeters at the center of the racket.

7.4 Discussion

The discussion section is organized as follows: first, global performances of the robotic system with a collaborative user are discussed (section 4.1). Secondly, the performances of the modules of the algorithm are analyzed, including the ball prediction module, the robot planner and the control structure (section 4.2). Finally, limitations and future works are discussed (section 4.3).

7.4.1 Global performance with a collaborative user

The first main result is that our system based on a desktop robotic arm, using the developed algorithm can play small rallies with a collaborative user. As shown in Table 3, the robot achieved an average return rate of 71.3% of the balls to the user, and up to 81.4% with optimized return parameters, allowing for short rallies averaging 4-5 returns to the user. The robot also achieved a record rally of 32 consecutive hits. In addition, the robot was able to hit 94.7% of the balls during

rallying. The results are a bit inferior compared to results with similar configurations with 7-DOF industrial robots, WAM [9] and Agilus [10] (return rate of 87% and 88%, respectively). Most balls that were not successfully returned either hit the net, went too long, or missed the table, primarily due to incorrect racket speed during the hit. Two main physical limitations of our desktop robotic arm can explain these errors. First, the maximum speed of the motors, which ranges from 30 to 77 RPM may prevent the robot from achieving the required speed during hits, even when the planner identifies the configuration requiring the smallest speeds. Secondly, due to the arm architecture with 5 serial DOF, it is difficult for the robotic arm to perform straight-line movements, as in [10], [30]. As a result, the racket often has a rotational speed during the hit, increasing the return error for the same placement error.

Failure to contact the ball are mainly caused by two situations: either an unexpected bounce of the ball, or a last-minute robot configuration change, when the hitting point is near the edge of the robot workspace. In most cases, these two situations are interrelated, and are caused by the architecture of the robot. In fact, as the robot is a 5-DOF serial robotic arm, it lacks redundant DOF to adapt quickly to configuration changes, resulting in a discontinuous "joint-to-Cartesian" workspace. This issue could be addressed by modifying the system architecture, to specialize it for table tennis applications. All current high-performing systems have also been adapted for table tennis. In [4], a planar 2-DOF moving base is used to quickly adapt to changes in the ball trajectory. In [30], the structure of the delta robot provides a continuous workspace, allowing for quick trajectory adjustments. In [10], the robot redundant DOF and larger size enable it to hit all the balls in a same vertical plan, always using the same configuration.

Table 4 also shows the performances of the system under non-rallying conditions. The robot successfully returned up to 21 consecutive balls, with an average return rate of 80.2 %. This return rate even increased to 89.4% with optimized return parameters. The difference in performance between rallying and non-rallying conditions can be largely attributed to the fact that the initial throw is made farther from the robot compared to the hits during a rally. In addition, the robot hit 97.7% of the balls and even 100% of the balls thrown during tests with optimized return parameters. These high hitting and return rates, despite the non-optimal physical architecture are the result of the changes made in the algorithm to evaluate the largest number of configurations for returning the ball.

7.4.2 Module performances

7.4.2.1 Ball trajectory prediction

The ball prediction module predicts the ball's trajectory with an RMSE of 65 mm within 1.8 ms. The computation time is faster than the camera's frequency (100 Hz), allowing for continuous updates and corrections in predictions as the ball approaches the robot. Compared to the optimal Hybrid³ method presented in [45], which combines GRU-TAE, TAE, and model-based prediction with spin estimation, our method has a 25% higher error. However, it is 29.4 times faster, making it more suitable for real-time table tennis applications.

While our trajectory prediction method is not the most accurate, the parameter optimization performed before play—using the machine learning algorithms from [45]—along with the low computation time compensates for the accuracy loss. This enables real-time corrections, even after the ball rebounds, to account for speed changes due to spin. Nonetheless, incorporating spin estimation and GRU-TAE on a faster computing platform could further improve accuracy.

Despite the 25% degradation in accuracy compared to the optimal prediction method from [45], the RMSE during initial ball predictions remains below the radius of a table tennis racket, approximately 75mm. This ensures sufficient accuracy without requiring configuration changes between the initial trajectory prediction and the final prediction before the hit, while real-time updates allow for small corrections just before impact.

7.4.2.2 Robot planner

The robot planner can generate a robot trajectory from the ball prediction within 7 ms, which is faster than the camera’s sampling frequency — 100Hz. This enables the robot to quickly adapt to changes in the ball trajectory prediction. In redundant and/or Cartesian system capable of straight movements, the hitting point is selected before computing the hitting configuration, such as on a horizontal or vertical plane [4], [10], [30]. This approach eliminates the need to explore multiple configurations, allowing for almost instantaneous trajectory generation. However, this simplification cannot be applied to our desktop system. Here, the main challenge is the lack of redundancy and a continuous workspace, requiring the computation of multiple configurations — racket orientations and hitting points. Thanks to steps of filtering which eliminate most of the

solutions at each step and matrix calculations, the computation time remains below 10 ms, while exploring thousands of configurations.

In addition, although quintic polynomials for trajectory generation reduce robustness compared to linear trajectories, they help minimize constraints on motors with limited speed and torque, enabling them to track trajectories that return the ball to the user.

7.4.2.3 Robot control

The proposed control structure based on a PD controller with neural network feedforward compensation, from [34], showed an accuracy of 0.88 degrees in the last three joints. When applied to our 5-DOF desktop robotic arm, this results in a maximum error of 3.1 centimeters, which is still less than the radius of the racket — 75 mm. The main interest of this control structure is that it requires no information about the dynamic model [34]. In systems using industrial robotic arms, the control box is typically internal to the arm [4], [10], [30], with the dynamic model well-known and included in the control algorithm. However, with desktop robotic arms, with 3D-printed components, obtaining precise dynamic parameters is more challenging, making this method particularly suited for our system. In addition, the feedforward correction is computed in real time — i.e. in less than 1ms, which gives enough time for other modules of the algorithm.

7.4.2.4 Limitations and future works

The main limitation of this study is that the robot's performance was evaluated with a collaborative user who provided ball throws within the workspace of the robotic arm as much as possible, with minimal spin and speed. This allowed the system sufficient time to react and move to the desired configuration to return the ball. These limitations are related to the robot's architecture. Using a 5-DOF serial desktop robotic arm without a redundant joint makes it difficult to execute linear movements near the hit, reducing the system's robustness. In addition, the fixed base limits the workspace of the robot, compared to those mounted on a movable basis [4]. Future work should explore desktop architectures better suited for table tennis, enabling easier linear movements or providing a workspace that covers the entire table as in [30]**.** Furthermore, incorporating mechanical components capable of higher dynamics would enhance performance. Such an architecture would allow the algorithm to optimize hit parameters for maximizing robustness rather than iterating over all solutions to find the least demanding option for the motors.

Regarding the algorithm, the main limitation is that the spin of the ball is not directly considered when determining the required racket speed during a hit. To compensate for this, the robot updates itself after the ball's rebound to adapt to changes in the ball trajectory.

7.5 Conclusion

The objective of this paper was to develop an algorithm for a collaborative table tennis system based on a desktop robotic arm to demonstrate the achievable performances by desktop robotic arms integrating artificial intelligence.

The developed algorithm consisted of four modules, each optimized using optimized methods from state-of-the-art, specifically adapted for playing table tennis with a desktop robotic arm: the vision system, ball trajectory prediction, robot planner — including the computation of optimal hitting configurations, inverse kinematics, and trajectory planning — and a control structure for trajectory tracking. The results were demonstrated through real table tennis play with a collaborative user.

The main results were as follows: 1. The developed algorithm allows the 5-DOF desktop system to successfully return 71.4% of the balls to the user overall, and up to 81.3% with optimized return parameters, achieving small rallies averaging 4-5 returns with a collaborative user. This result is quite close to results from literature, using 7-DOF industrial serial robots. 2. The system hits 94.7% of the incoming balls, with misses typically resulting from bad bounces or unavoidable last-minute configuration changes due to the robot architecture. 3. Computation times of all the modules of the algorithm were optimized, enabling the evaluation of an average of 2500 hitting configurations in less than 10 ms, allowing to quickly adapt to perturbations in the ball trajectory, while choosing the less demanding configuration for the motors.

These results are important because they demonstrate the potential of using desktop robotic arms with algorithms integrated with artificial intelligence in high-speed collaboration with humans. For table tennis, future work will focus on applying the developed table tennis algorithm to a desktop robotic architecture better suited for table tennis. Further than table tennis, future works will also extend the use of desktop robots to other applications and further improve their performance.

7.6 References

[1] Z. Fu, T. Z. Zhao, and C. Finn, “Mobile ALOHA: Learning Bimanual Mobile Manipulation with Low-Cost Whole-Body Teleoperation,” Jan. 04, 2024, *arXiv*. Accessed: Aug. 26, 2024. [Online]. Available: http://arxiv.org/abs/2401.02117

[2] J. Wu *et al.*, “Tidybot: Personalized robot assistance with large language models,” *Autonomous Robots*, vol. 47, no. 8, pp. 1087–1102, 2023.

[3] C. Li, M. Vlastelica, S. Blaes, J. Frey, F. Grimminger, and G. Martius, “Learning agile skills via adversarial imitation of rough partial demonstrations,” in *Conference on Robot Learning*, PMLR, 2023, pp. 342–352. Accessed: Aug. 26, 2024. [Online]. Available: https://proceedings.mlr.press/v205/li23b.html

[4] D. B. D’Ambrosio *et al.*, “Achieving Human Level Competitive Robot Table Tennis,” Aug. 09, 2024, *arXiv*. Accessed: Aug. 26, 2024. [Online]. Available: http://arxiv.org/abs/2408.03906

[5] J. Billingsley, “Robot ping pong, Practical Computing.” Cambridge: MIT press, 1983.

[6] R. L. Anderson, *A robot ping-pong player: experiment in real-time intelligent control*. MIT press, 1988. Accessed: Jun. 19, 2024. [Online]. Available: https://dl.acm.org/doi/abs/10.5555/42334

[7] H. Hashimoto, F. Ozaki, and K. Osuka, “Development of a pingpong robot system using 7 degrees of freedom direct drive arm,” in *IECON’87: Industrial Applications of Robotics & Machine Vision*, SPIE, 1987, pp. 608–615.

[8] J. Knight and D. Lowery, “Pingpong-playing robot controlled by a microcomputer,” *Microprocessors and Microsystems*, vol. 10, no. 6, pp. 332–335, 1986.

[9] K. Mülling, J. Kober, O. Kroemer, and J. Peters, “Learning to select and generalize striking movements in robot table tennis,” *The International Journal of Robotics Research*, vol. 32, no. 3, pp. 263–279, Mar. 2013, doi: 10.1177/0278364912472380.

[10] J. Tebbe, Y. Gao, M. Sastre-Rienietz, and A. Zell, “A Table Tennis Robot System Using an Industrial KUKA Robot Arm,” in *Pattern Recognition*, vol. 11269, T. Brox, A. Bruhn, and M. Fritz, Eds., in Lecture Notes in Computer Science, vol. 11269. , Cham: Springer International Publishing, 2019, pp. 33–45. doi: 10.1007/978-3-030-12939-2_3.

[11] D. Büchler, S. Guist, R. Calandra, V. Berenz, B. Schölkopf, and J. Peters, “Learning to Play Table Tennis From Scratch Using Muscular Robots,” *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3850–3860, Dec. 2022, doi: 10.1109/TRO.2022.3176207.

[12] A. Nakashima, Y. Ogawa, C. Liu, and Y. Hayakawa, “Robotic table tennis based on physical models of aerodynamics and rebounds,” in *International Conference on Robotics and Biomimetics*, IEEE, 2011, pp. 2348–2354. Accessed: Aug. 26, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6181649

[13] Y. Gao, J. Tebbe, J. Krismer, and A. Zell, “Markerless racket pose detection and stroke classification based on stereo vision for table tennis robots,” in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pp. 189–196. Accessed: Aug. 26, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8675584

[14] P. Blank, B. H. Groh, and B. M. Eskofier, “Ball speed and spin estimation in table tennis using a racket-mounted inertial sensor,” in *ACM International Symposium on Wearable Computers*, Sep. 2017, pp. 2–9. doi: 10.1145/3123021.3123040.

[15] T. Gossard, J. Krismer, A. Ziegler, J. Tebbe, and A. Zell, “Table tennis ball spin estimation with an event camera,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 3347–3356. Accessed: Aug. 26, 2024. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2024W/CVsports/html/Gossard_Table_Tennis_Ball_Spin_Estimation_with_an_Event_Camera_CVPRW_2024_paper.html

[16] K. Muelling, A. Boularias, B. Mohler, B. Schölkopf, and J. Peters, “Learning strategies in table tennis using inverse reinforcement learning,” *Biological cybernetics*, vol. 108, pp. 603–619, 2014.

[17] Z. Wang *et al.*, “Probabilistic movement modeling for intention inference in human–robot interaction,” *The International Journal of Robotics Research*, vol. 32, no. 7, pp. 841–858, Jun. 2013, doi: 10.1177/0278364913478447.

[18] Z. Wang, A. Boularias, K. Mülling, B. Schölkopf, and J. Peters, “Anticipatory action selection for human–robot table tennis,” *Artificial Intelligence*, vol. 247, pp. 399–414, 2017.

[19] K. Muelling, J. Kober, and J. Peters, “Learning table tennis with a Mixture of Motor Primitives,” in *2010 10th IEEE-RAS International Conference on Humanoid Robots*, pp. 411–416. doi: 10.1109/ICHR.2010.5686298.

[20] O. Koç, G. Maeda, and J. Peters, “Online optimal trajectory generation for robot table tennis,” *Robotics and Autonomous Systems*, vol. 105, pp. 121–137, Jul. 2018, doi: 10.1016/j.robot.2018.03.012.

[21] Y. Zhu, Y. Zhao, L. Jin, J. Wu, and R. Xiong, “Towards high level skill learning: Learn to return table tennis ball using monte-carlo based policy gradient method,” in *2018 IEEE international conference on real-time computing and robotics (RCAR)*, pp. 34–41. Accessed: Aug. 21, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8621776

[22] J. Tebbe, L. Krauch, Y. Gao, and A. Zell, “Sample-efficient reinforcement learning in robotic table tennis,” in *2021 IEEE international conference on robotics and automation (ICRA)*, pp. 4171–4178. Accessed: Aug. 21, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9560764

[23] Y. Huang, B. Schölkopf, and J. Peters, “Learning optimal striking points for a ping-pong playing robot,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 4587–4592. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7354030

[24] T. Ding *et al.*, “GoalsEye: Learning High Speed Precision Table Tennis on a Physical Robot,” Oct. 13, 2022, *arXiv*. doi: 10.48550/arXiv.2210.03662.

[25] C. Liu, Y. Hayakawa, and A. Nakashima, “Racket Control for a Table Tennis Robot to Return a Ball,” *SICE Journal of Control, Measurement, and System Integration*, vol. 6, no. 4, pp. 259–266, Jul. 2013, doi: 10.9746/jcmsi.6.259.

[26] S. W. Abeyruwan *et al.*, “i-sim2real: Reinforcement learning of robotic policies in tight human-robot interaction loops,” in *Conference on Robot Learning*, PMLR, 2023, pp. 212–224. Accessed: Aug. 26, 2024. [Online]. Available: https://proceedings.mlr.press/v205/abeyruwan23a.html

[27] M. Matsushima, T. Hashimoto, and F. Miyazaki, “Learning to the robot table tennis task-ball control & rally with a human,” in *2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme-System Security and Assurance (Cat. No. 03CH37483)*, pp. 2962–2969. Accessed: Aug. 21, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1244342/

[28] Y. Sun, R. Xiong, Q. Zhu, J. Wu, and J. Chu, “Balance motion generation for a humanoid robot playing table tennis,” in *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pp. 19–25. Accessed: Aug. 21, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6100826

[29] L. Chen, R. Paleja, and M. Gombolay, “Learning from Suboptimal Demonstration via Self-Supervised Reward Regression,” in *Proceedings of the 2020 Conference on Robot Learning*, PMLR, pp. 1262–1277. Accessed: Aug. 21, 2024. [Online]. Available: https://proceedings.mlr.press/v155/chen21b.html

[30] A. Kyohei, N. Masamune, and Y. Satoshi, “The ping pong robot to return a ball precisely,” *Omron Technics*, vol. 51, pp. 1–6, 2020.

[31] D. B. D’Ambrosio *et al.*, “Robotic Table Tennis: A Case Study into a High Speed Learning System,” Sep. 06, 2023. doi: 10.15607/RSS.2023.XIX.006.

[32] Y. Zhang, W. Wei, D. Yu, and C. Zhong, “A tracking and predicting scheme for ping pong robot,” *Journal of Zhejiang University SCIENCE C*, vol. 12, no. 2, pp. 110–115, 2011.

[33] L. Acosta, J. J. Rodrigo, J. A. Mendez, G. N. Marichal, and M. Sigut, “Ping-pong player prototype,” *IEEE robotics & automation magazine*, vol. 10, no. 4, pp. 44–52, 2003.

[34] B. Toussaint and M. Raison, “Design of Minimal Model-Free Control Structure for Fast Trajectory Tracking of Robotic Arms,” *Applied Sciences*, vol. 14, no. 18, p. 8405, 2024.

[35] L. R. Soares and V. H. C. Alcalde, “An educational robotic workstation based on the rhino XR4 robot,” in *Proceedings. Frontiers in Education. 36th Annual Conference*, IEEE, 2006, pp. 7–12. Accessed: Dec. 09, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4116936/

[36] O. N. Sahin, E. Uzunoglu, E. Tatlicioglu, and M. I. C. Dede, “Design and development of an educational desktop robot R^3 D,” *Comp Applic In Engineering*, vol. 25, no. 2, pp. 222–229, Mar. 2017, doi: 10.1002/cae.21792.

[37] D. Zhou, R. Jia, and M. Xie, “Mirobot: A Low-Cost 6-DOF Educational Desktop Robot,” in *Robotics in Education*, vol. 1359, M. Merdan, W. Lepuschitz, G. Koppensteiner, R. Balogh, and D. Obdržálek, Eds., in Advances in Intelligent Systems and Computing, vol. 1359. , Cham: Springer International Publishing, 2022, pp. 189–200. doi: 10.1007/978-3-030-82544-7_18.

[38] I. Chavdarov, V. Nikolov, B. Naydenov, and G. Boiadjiev, “Design and control of an educational redundant 3D printed robot,” in *International Conference on Software,*

Telecommunications and Computer Networks (SoftCOM), IEEE, 2019, pp. 1–6. Accessed: Dec. 09, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8903825/

[39] A. Matić, P. Valerjev, and A. Gomez-Marin, "Hierarchical control of visually-guided movements in a 3D-printed robot arm," *Frontiers in Neurorobotics*, vol. 15, p. 755723, 2021.

[40] "Motive - In Depth," OptiTrack. Accessed: Oct. 28, 2024. [Online]. Available: http://optitrack.com/software/motive/index.html

[41] Robotis, "ROBOTIS e-Manual," ROBOTIS e-Manual. Accessed: Nov. 04, 2024. [Online]. Available: https://emanual.robotis.com/docs/en/dxl/x/xm540-w270/

[42] B. Toussaint and M. Raison, "Real-time trajectory prediction of a ping-pong ball using a GRU-TAE," *Appl Intell*, vol. 55, no. 5, p. 339, Apr. 2025, doi: 10.1007/s10489-024-06204-4.

[43] T. Gossard, J. Tebbe, A. Ziegler, and A. Zell, "SpinDOE: A ball spin estimation method for table tennis robot," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 5744–5750. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10342178

[44] S. Gomez-Gonzalez, S. Prokudin, B. Schölkopf, and J. Peters, "Real time trajectory prediction using deep conditional generative models," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 970–976, 2020.

[45] B. Toussaint and M. Raison, "Hybrid approach for ball trajectory prediction in table tennis: combining machine learning and physical models," *Engineering Applications of Artificial Intelligence*, Oct. 2024.

[46] S. M. Toolbox, "Matlab," *Mathworks Inc*, p. 20, 1993.

[47] "scikit-learn: machine learning in Python — scikit-learn 1.5.0 documentation." Accessed: Jun. 25, 2024. [Online]. Available: https://scikit-learn.org/stable/

[48] ITTF, "ITTF Handbook 2022." 2022. [Online]. Available: https://documents.ittf.sport/sites/default/files/public/2022-02/ITTF_HB_2022_clean_v1_0.pdf

CHAPITRE 8 DISCUSSION GÉNÉRALE

L'objectif principal de cette thèse était de développer un système collaboratif de tennis de table basé sur un bras robotique de bureau, afin de démontrer les performances atteignables par des bras robotiques de bureau, intégrant l'intelligence artificielle. Cet objectif a été complété en répondant aux quatre sous-objectifs suivants :

SO1 : Développer une méthode de prédiction de trajectoire basée sur un modèle seq2seq récurrent présentée au chapitre 4.

SO2 : Concevoir une méthode de prédiction de trajectoires, combinant l'IA avec des approch-es traditionnelles basées sur le modèle dynamique, présentée au chapitre 5.

SO3 : Développer une structure de contrôle minimale, sans connaissance préalable du modèle dynamique en utilisant des réseaux de neurones, présentée dans le chapitre 6.

SO4 : Développer un algorithme pour le tennis de table, adapté spécifiquement aux contraintes des bras robotiques de bureau. Ce système a été présenté au chapitre 7.

Ce chapitre présente une discussion générale sur l'atteinte des objectifs de cette thèse, expose les limites du projet et propose des pistes pour des travaux futurs.

8.1 Développement d'une méthode de prédiction de trajectoire basée sur un modèle seq2seq récurrent

Les travaux présentés dans les chapitres 4 et 5 ont abordé le problème de la prédiction de trajectoire de la balle (SO1 et SO2). En particulier, les méthodes issues de la revue de littérature pour la prédiction des séries temporelles ou utilisées dans les systèmes actuels ont été résumées dans la Figure 4.1. Ces méthodes ont ensuite été comparées de manière équitable : les méthodes d'apprentissage machine [13], [108], [135], récursives, et « seq2seq », et les méthodes basées sur les modèles dynamiques [12], [106], [110]. Cette comparaison, la première d'une telle ampleur, a permis d'identifier les méthodes offrant la meilleure précision, ainsi que leurs avantages et inconvénients.

8.1.1 Prédiction de trajectoire basée sur un modèle seq2seq récurrent

Les travaux du chapitre 4 ont montré que l'utilisation des modèles « seq2seq » et en particulier de l'architecture des autoencodeurs de trajectoire (TAE), permet d'obtenir la meilleure précision

parmi les méthodes d'apprentissage machine. Ces modèles permettent une amélioration moyenne de 44% de la précision de la prédiction par rapport aux modèles récursifs (Figure 4.5). Cette supériorité est due à l'absence d'erreurs cumulatives, illustrées dans les Figures 4.6 à 4.8, et principale limite des modèles récursifs. Ces résultats confirment les conclusions de [108]. Toutefois, les modèles récursifs se sont révélés plus précis pour les prédictions à court terme (moins de 0,1 s).

Sur la base de ces résultats, les méthodes identifiées ont été optimisées. Le TAE existant [108] a été adapté pour la prédiction de séries temporelles en intégrant des couches récursives dans l'architecture des réseaux de neurones. Un GRU-TAE a ainsi été utilisé pour la première fois dans la prédiction des trajectoires de balles de tennis de table, atteignant une erreur quadratique moyenne (RMSE) de 48mm, soit une amélioration de 14% par rapport au TAE de [108] (Figure 4.5).

8.1.2 Prédiction de trajectoire basée sur un modèle seq2seq récurrent

Les travaux du chapitre 5, et notamment le Tableau 5.3 ont mis en évidence l'importance d'identifier avec précision les paramètres dynamiques pour réaliser des prédictions à partir des équations du modèle dynamique.

Le processus d'identification des paramètres du modèle dynamique a été simplifié grâce à l'intégration de méthodes d'apprentissage machine. Des modèles différentiables ont été utilisés pour optimiser des paramètres dynamiques à partir de trajectoires réelles de la balle. Les résultats, obtenus à la fois en simulation et avec des données expérimentales, sont présentés dans le Tableau 5.4. Ils montrent que cette approche améliore la précision des prédictions de 42% par rapport à l'utilisation des paramètres disponibles dans la littérature [38]. Cependant, ces résultats doivent être nuancés. Pour réduire la complexité des calculs, le système repose sur des caméras infrarouges nécessitant l'utilisation d'un ruban réfléchissant fixé sur la balle. Le ruban modifie le modèle dynamique de la balle et peut générer des faux rebonds.

Une fois les paramètres optimisés, une méthode inspirée de [12] consistant à estimer le spin contenu dans la balle à partir de l'analyse de son impact sur la trajectoire a été implémenté. Les résultats ont démontré une précision satisfaisante avec une RMSE de 58mm pour des lancers expérimentaux (Figure 5.2) tout en limitant la complexité des calculs, correspondant à une diminution de 9% de la RMSE par rapport au GRU-TAE.

Enfin, le chapitre 5 introduit une méthode hybride combinant le GRU-TAE, le TAE et la prédiction par modèle dynamique. Cette approche réduit l'erreur moyenne des prédictions de 16 % supplémentaires (Figure 5.2) et montre l'intérêt de combiner les méthodes traditionnelles basées sur les modèles dynamique et les méthodes d'apprentissage machine. Cette méthode prometteuse pourra être optimisée davantage lors de travaux futurs, en prenant en compte l'incertitude associée à chaque approche à les différentes étapes de la prédiction.

8.2 Structure de contrôle minimale

Dans le chapitre 6, une structure de contrôle minimale pour le suivi de trajectoires rapides a été présentée (SO3). Le principal intérêt de ce travail réside dans le fait qu'elle ne requiert aucune connaissance préalable du modèle dynamique. En effet, un contrôleur proportionnel-dérivé (PD) en position, reconnu pour sa stabilité et sa simplicité d'utilisation [185] est renforcé avec un réseau de neurones capable d'estimer efficacement les systèmes dynamiques. La structure développée consiste à apprendre la réponse du contrôleur PD via un réseau de neurones, puis de l'utiliser dans une boucle de contrôle externe comme une compensation feedforward, permettant ainsi de prédire et de compenser les erreurs de suivi. L'architecture détaillée est présentée à la Figure 6.1.

Bien que cette approche ait déjà été envisagée dans des travaux précédents [162] [163], l'étude présentée au chapitre 6 constitue la première comparaison systématique de différentes variantes de cette structure de contrôle (INN et GNN) et l'optimisation des hyperparamètres associés. À partir des résultats obtenus, un guide pratique – présenté dans le Tableau 6.5 – a été élaboré pour permettre une utilisation optimale de cette méthode sans recourir à des essais fastidieux. Le code utilisé pour cette étude a également été partagé [186].

Les résultats expérimentaux obtenus à la fois sur des systèmes réels et simulés ont démontré une précision satisfaisante, avec une RMSE inférieure à 2° et une grande robustesse. Ces performances sont résumées dans le Tableau 6.4.

Au-delà de l'application pour le tennis de table, la méthode de contrôle hybride développée est applicable pour des applications variées. En particulier, cette méthode est adaptée à l'ensemble des applications mettant en jeu des trajectoires à grande amplitude et haute dynamique, sur des systèmes où il est difficile d'obtenir un modèle dynamique avec précision. Les guidelines fournies

dans le Chapitre 6 permettant d'optimiser les performances sont également valables quelle que soit l'application.

Si le contrôleur PID n'est pas adapté au suivi de trajectoires à haute dynamique impliquant de grandes amplitudes de mouvement, son intégration intelligente, en remplacement du contrôleur PD dans la structure de contrôle pourrait offrir des améliorations. Par exemple, l'ajout d'une composante intégrale adaptative pourrait permettre d'augmenter la précision pour des trajectoires plus lentes. Cette question pourra être explorée dans le cadre de futurs travaux.

8.3 Algorithme pour le tennis de table

Le chapitre 7 a présenté le développement d'un algorithme adapté spécifiquement aux contraintes des bras robotiques de bureau, ainsi que son intégration sur un bras robotique de bureau (SO4). Bien que plusieurs robots joueurs de tennis de table ont déjà été développés, ce système est, à notre connaissance, le premier robot personnel de bureau capable de réaliser des échanges de tennis de table avec un utilisateur .

Le bras robotique utilisé est un modèle à cinq degrés de liberté, développé par notre laboratoire, pour un coût en matériel inférieur à 3500 US$ grâce notamment à l'impression 3D. Ce coût est nettement inférieur au coût des cobots industriels habituellement employés – plus de 50000$ pour le KUKA Agilus [187], 25000$ pour l'IRB1100 de ABB [13].

L'absence de DDL redondants, combiné aux limites physiques des moteurs – notamment en couple et en vitesse – impose également d'adapter l'algorithme de contrôle. Ce dernier explore davantage de solutions afin de trouver celle qui minimise les efforts requis par les moteurs, comme détaillé dans la section 7.2.4.

Pour répondre à ces contraintes, les modules de l'algorithme ont donc été optimisés, tout en maintenant un temps de calcul très faible – environ 7ms, selon le Tableau 7.6 –. Les méthodes développées dans les chapitres 4 à 6 pour le contrôle et la prédiction de trajectoire de la balle ont été intégrées afin d'optimiser les performances. Les autres étapes de l'algorithme ont été implémentées à l'aide des techniques les plus performantes issues de l'état de l'art [12]. Enfin, l'utilisation de caméras infrarouges pour la détection de la balle a permis de réduire la charge de calcul.

Les capacités du bras robotique ont été validées expérimentalement par des tests réalisés face à un utilisateur humain, dans le cadre d'échanges libres – sans imposer un lancer spécifique de l'utilisateur. Ces tests réalisés dans des environnements variés – conditions d'éclairage, positionnement des caméras – et présentés dans les Tableaux 7.3 et 7.4 montrent la capacité du système à réaliser des échanges de tennis de table. Un taux de renvoi moyen de 71.3%, et même 81.4% en optimisant certains paramètres, comme la position de la base du robot, ou le point visé pour le renvoi a été mesuré. Ce taux correspond à une moyenne d'environ 4 à 5 allers-retours de la balle par échange.

Cependant, comme discuté dans les chapitres précédents, la limitation majeure réside dans l'architecture physique du robot. L'utilisation d'un bras sériel à cinq DDL pour le tennis de table est un vrai défi étant donné l'absence de DDL redondants. Cette caractéristique complique la réalisation de trajectoires linéaires, plus robustes pour le renvoi de la balle. Comme mentionné dans la revue de littérature, les autres systèmes capables de jouer au tennis de table disposent généralement de DDL redondants [11], [16], ou d'une architecture permettant ces mouvements linéaires, telles que des robots parallèles [12] ou des actionneurs linéaires [13]. Cette limitation, conjuguée aux limites de vitesse des moteurs, explique la majorité des échecs du robot à renvoyer la balle. Ainsi, la plupart des balles non renvoyées à l'utilisateur terminent dans le filet – vitesse maximale des moteurs trop faible – ou à côté de la table – manque de robustesse dû à la vitesse de rotation de la raquette. Enfin, les échecs du robot à toucher la balle sont très largement dus à son espace de travail limité, et non continu. Une amélioration future consisterait à adapter l'architecture du robot ou à intégrer des actionneurs capables d'atteindre des vitesses maximales plus élevées, augmentant ainsi la réactivité du système.

8.4 Contributions de recherche

L'objectif principal de cette thèse était de démontrer le potentiel des bras robotiques personnels, imprimés en 3D, en développant un système robotique capable de jouer au tennis de table, basé sur le prototype développé dans le laboratoire du professeur Maxime Raison. Les contributions de cette recherche ont permis l'atteinte de cet objectif principal et des objectifs spécifiques définis. Ces contributions ont été publiées (ou soumises pour publication) dans des journaux révisés par des pairs.

La Tableau 8.1 récapitule les principales contributions de cette thèse, en les associant aux objectifs spécifiques (OS) et aux publications correspondantes.

Tableau 8.1 : Contributions de recherche

Contribution	Objectif spécifique	Publications associées
Nouvelle connaissance : Les méthodes seq2seq offrent une meilleure précision pour la prédiction des trajectoires, en particulier à long terme, par rapport aux méthodes récursives.	OS1	A
Nouvelle méthode : Utilisation d'un GRU-TAE pour la prédiction des trajectoires de balles de tennis de table.	OS1	A
Nouvelle méthode : Exploitation des modèles différentiables pour optimiser les paramètres dynamiques de la balle.	OS2	B
Nouvelle méthode : Combinaison d'un GRU-TAE, d'un TAE et des équations dynamiques pour améliorer la précision des prédictions.	OS2	B
Nouvelle méthode : Développement d'une structure de contrôle pour le suivi de trajectoires à haute dynamique, sans connaissance préalable du modèle dynamique.	OS3	C
Nouveau contenu : Création d'un guide en cinq étapes pour concevoir systématiquement la structure de contrôle développée.	OS3	C
Nouvel algorithme : Intégration des méthodes développées dans un algorithme adapté aux contraintes des bras robotiques de bureau.	OS4	D
Nouveau système : Développement du premier système joueur de tennis de table basé sur un bras robotique personnel de bureau, à cinq DDL.	OS4	D

8.5 Retour sur le projet

La motivation principale de ce projet était de démontrer que l'IA et l'apprentissage machine, combinés aux nouvelles techniques d'impression 3D, permettent de réduire l'écart de performance entre les systèmes robotiques de bureau – à faible coût – et les bras industriels, notamment en matière de dextérité, rapidité et précision.

Les résultats obtenus au cours de cette recherche (taux de renvoi moyen de 71.3%, et même 81.4% en optimisant certains paramètres) ont démontré qu'il est possible de réaliser des tâches complexes exigeant un niveau de performance élevé, tel que le jeu du tennis de table, en utilisant un bras robotique de bureau, imprimé en 3D. Les tests expérimentaux réalisés dans des conditions variées – éclairage, positions des caméras, structures de maintien – ont montré l'adaptabilité du système. Ces travaux ont également souligné l'apport des techniques d'apprentissage dans divers domaines, en simplifiant les processus de développement tout en maintenant des performances élevées.

Les avancées réalisées dans le cadre de cette thèse pour le tennis de table sont applicables à d'autres contextes robotiques, tels que la robotique d'assistance. Si leur usage dans les environnements domestiques se développe [3], nous avons vu que leur coût élevé et leur manque d'adaptabilité constituaient un frein majeur à leur adoption [3]. Le bras robotique utilisé dans ce projet, développé dans le laboratoire du professeur Maxime Raison, est un bras d'assistance personnel conçu pour réduire les coûts matériels (moins de 3 000 $). Les performances présentées dans le chapitre 7 ont montré que ce bras robotique possède des capacités lui permettant un développement potentiel en réadaptation ou en assistance personnelle. D'autres travaux en robotique d'assistance ont d'ailleurs été menés à partir de cette au sein du laboratoire [188], [189]. De plus, la structure de contrôle présentée au Chapitre 6 est applicable à d'autres systèmes robotiques en appliquant les guidelines fournies. Ceci permettrait d'étendre les méthodes de prédiction de trajectoire à des applications en dehors du tennis de table.

Pour une utilisation plus accessible à un particulier, il serait également préférable d'inclure un traitement d'images robuste pour calculer la position de la balle en temps-réel dans un environnement non contrôlé. Cela permettrait d'éviter l'utilisation d'un ruban réfléchissant sur la balle, pouvant altérer sa dynamique.

Enfin, les résultats obtenus ont mis en lumière des limitations liées au positionnement des degrés de liberté pour le tennis de table. Ces limitations concernent en particulier la réalisation de trajectoires linéaires. Le compromis entre le niveau de performance des systèmes et leur coût reste inévitable. Cependant, si augmenter directement la puissance des moteurs sans accroître les coûts reste un défi, il est néanmoins possible — et indispensable — d'optimiser l'architecture en fonction de l'application visée. Dans le cadre du tennis de table, pour augmenter la vitesse maximale du système tout en maintenant un coût modéré, il serait utile minimiser la masse des parties mobiles. Si les robots parallèles ont déjà été utilisés, les robots à câbles, qui possèdent un grand espace de travail et des masses mobiles réduites pourraient être des candidats sérieux pour le tennis de table. Cependant, la synchronisation du mouvement des différents moteurs afin de maintenir la tension dans les câbles est un point critique, particulièrement au vu de la rapidité nécessaire dans les trajectoires. De plus, la structure nécessaire à un mouvement dans l'espace est assez imposante. Une version hybride planaire, moins imposante, avec deux moteurs pour orienter la raquette pourrait par exemple offrir une solution. Plus largement, la réduction des coûts des systèmes

robotiques nécessaire à leur démocratisation dans la société passera donc également par le développement d’architectures plus adaptables et plus polyvalentes.

CHAPITRE 9 CONCLUSION ET RECOMMANDATIONS

L'objectif de cette thèse était de développer un système collaboratif de tennis de table, basé sur le prototype du laboratoire du professeur Maxime Raison afin de démontrer les performances atteignables par des bras robotiques de bureau, intégrant l'intelligence artificielle. L'algorithme développé pour ce système inclut les sept principales étapes pour le jeu du tennis de table. La détection de la balle, la prédiction de sa trajectoire, l'identification des points de frappe possibles, le choix du point de frappe optimal, le processus de cinématique inverse, la génération de trajectoire du robot, et la structure de contrôle permettant le suivi de cette trajectoire.

La première étape consiste à utiliser des caméras infrarouges un adhésif réfléchissant fixé sur la balle pour permettre la détection robuste et précise.

La deuxième concerne la prédiction de la trajectoire de la balle. Pour cela, les méthodes issues de la littérature, notamment l'apprentissage machine et les équations du modèle dynamique ont été implémentées et comparées pour la première fois sur un pied d'égalité. Les résultats ont révélé que les approches les plus précises étaient l'autoencodeur de trajectoire (TAE) et l'utilisation des paramètres dynamiques avec estimation du spin à partir de son impact sur la trajectoire. Ces deux techniques ont été retravaillées et optimisées. Un GRU-TAE, incluant des couches récursives pour la prédiction de séries temporelles, a été développé, offrant une amélioration de 14 % de la précision des prédictions. Parallèlement, les équations du modèle dynamique ont été optimisées grâce à des modèles différentiables pour estimer les paramètres dynamiques, démontrant de meilleures performances par rapport aux estimations initiales.

Les résultats mettent en lumière la complémentarité de ces approches, et cette thèse propose de les combiner pour améliorer encore la précision. Si prendre la moyenne des prédictions permet de réduire l'erreur moyenne de 16% supplémentaires, des travaux futurs pourront identifier des façons plus efficaces de combiner ces deux méthodes, en prenant en compte leurs incertitudes respectives.

La troisième étape a été le développement d'une structure de contrôle permettant le suivi de trajectoires rapides et à haute dynamique, sans connaissance préalable du modèle dynamique. L'architecture proposée consiste à combiner un contrôleur PD en position avec un réseau de neurones. Ce dernier apprend la réponse du PD, et est ensuite utilisé dans une boucle externe pour fournir une compensation feedforward. Les résultats obtenus sur des systèmes simulés et réels ont

montré que l'erreur quadratique moyenne était inférieure à 2 degrés. Un guide en cinq étapes a été réalisé, permettant reproduire facilement la structure de contrôle tout en optimisant les résultats.

Les autres étapes de l'algorithme ont été implémentées avec les méthodes identifiées comme les plus performantes, provenant de la littérature.

Le système final développé est capable de réaliser des échanges avec un utilisateur collaboratif, renvoyant jusqu'à 81.4% en optimisant la position de la base du robot et le point visé lors du retour.

Le système final développé dans cette thèse requiert néanmoins un travail additionnel afin de permettre une utilisation plus large par les particuliers. En particulier, si l'architecture du bras robotique utilisé – sériel, à 5 DOF – est utile pour de nombreuses applications, notamment en réadaptation, telle que la saisie d'objets, elle n'est pas optimisée pour le tennis de table. En effet, l'architecture rend complexe la réalisation de trajectoires rectilignes par la raquette, ce qui est primordial pour augmenter la robustesse du système pour le tennis de table. Il sera donc judicieux de développer une autre architecture pour le robot permettant de réaliser des mouvements rectilignes plus facilement.

En conclusion, ce système représente une étape supplémentaire dans la démocratisation des bras robotiques personnels et de bureau. Les travaux menés ont permis de jouer au tennis de table, tâche qui constitue un réel défi depuis plusieurs décennies dans le domaine de la robotique du fait de sa complexité, avec un bras robotique de bureau. Ces résultats ont montré que l'intégration des méthodes d'apprentissage machine, en complément des méthodes traditionnelles basées sur les modèles dynamiques, permet d'améliorer les performances des robots personnels. Les perspectives de cette recherche incluent l'utilisation d'une architecture physique spécifiquement optimisée pour le tennis de table permettant de réaliser plus facilement les mouvements rectilignes ainsi que l'extension des applications des bras robotiques de bureau à d'autres domaines, tout en continuant à accroitre leurs performances.

RÉFÉRENCES

[1] P. Coutance, “Le marché de la robotique pourrait tripler d’ici 2030,” VIPress.net. Accessed: Sep. 13, 2024. [Online]. Available: https://vipress.net/le-marche-de-la-robotique-pourrait-tripler-dici-2030/

[2] “Thematic Research on Robotics,” Market Research Reports & Consulting | GlobalData UK Ltd. Accessed: Sep. 13, 2024. [Online]. Available: https://www.globaldata.com/store/report/robotics-theme-analysis/

[3] pxdraft, “Taille et part du marché des robots de réadaptation | Rapport, 2030.” Accessed: Sep. 13, 2024. [Online]. Available: https://www.kingsresearch.com/fr/rehabilitation-robots-market-319

[4] C. Taesi, F. Aggogeri, and N. Pellegrini, “COBOT applications—recent advances and challenges,” *Robotics*, vol. 12, no. 3, p. 79, 2023.

[5] I. Lee, “Service robots: a systematic literature review,” *Electronics*, vol. 10, no. 21, p. 2658, 2021.

[6] E. Prassler, M. E. Munich, P. Pirjanian, and K. Kosuge, “Domestic Robotics,” in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds., in Springer Handbooks. , Cham: Springer International Publishing, 2016, pp. 1729–1758. doi: 10.1007/978-3-319-32552-1_65.

[7] T. Palleja, M. Tresanchez, M. Teixidó, and J. Palacin, “Modeling floor-cleaning coverage performances of some domestic mobile robots in a reduced scenario,” *Robotics and Autonomous Systems*, vol. 58, no. 1, pp. 37–45, 2010.

[8] I. Vallivaara, J. Haverinen, A. Kemppainen, and J. Röning, “Magnetic field-based SLAM method for solving the localization problem in mobile robot floor-cleaning task,” in *2011 15th international conference on advanced robotics (ICAR)*, IEEE, 2011, pp. 198–203. Accessed: Feb. 28, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6088632/

[9] C. Jayawardena *et al.*, “Deployment of a service robot to help older people,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2010, pp. 5990–5995. Accessed: Feb. 28, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5649910/

[10] R. L. Andersson, *A robot ping-pong player*, vol. 988. MIT press Cambridge, Massachusetts, 1988. Accessed: Aug. 21, 2024. [Online]. Available: https://direct.mit.edu/books/monograph/4796/bookpreview-pdf/2428755

[11] J. Tebbe, Y. Gao, M. Sastre-Rienietz, and A. Zell, “A Table Tennis Robot System Using an Industrial KUKA Robot Arm,” in *Pattern Recognition*, vol. 11269, T. Brox, A. Bruhn, and M. Fritz, Eds., in Lecture Notes in Computer Science, vol. 11269. , Cham: Springer International Publishing, 2019, pp. 33–45. doi: 10.1007/978-3-030-12939-2_3.

[12] A. Kyohei, N. Masamune, and Y. Satoshi, “The ping pong robot to return a ball precisely,” *Omron Technics*, vol. 51, pp. 1–6, 2020.

[13] D. B. D'Ambrosio *et al.*, "Achieving Human Level Competitive Robot Table Tennis," Aug. 09, 2024, *arXiv*. Accessed: Aug. 26, 2024. [Online]. Available: http://arxiv.org/abs/2408.03906

[14] H. Hashimoto, F. Ozaki, and K. Osuka, "Development of a pingpong robot system using 7 degrees of freedom direct drive arm," in *IECON'87: Industrial Applications of Robotics & Machine Vision*, SPIE, 1987, pp. 608–615.

[15] J. Knight and D. Lowery, "Pingpong-playing robot controlled by a microcomputer," *Microprocessors and Microsystems*, vol. 10, no. 6, pp. 332–335, 1986.

[16] K. Mülling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *The International Journal of Robotics Research*, vol. 32, no. 3, pp. 263–279, Mar. 2013, doi: 10.1177/0278364912472380.

[17] D. Büchler, S. Guist, R. Calandra, V. Berenz, B. Schölkopf, and J. Peters, "Learning to Play Table Tennis From Scratch Using Muscular Robots," *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3850–3860, Dec. 2022, doi: 10.1109/TRO.2022.3176207.

[18] A. Nakashima, Y. Ogawa, C. Liu, and Y. Hayakawa, "Robotic table tennis based on physical models of aerodynamics and rebounds," in *International Conference on Robotics and Biomimetics*, IEEE, 2011, pp. 2348–2354. Accessed: Aug. 26, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6181649

[19] Y. Gao, J. Tebbe, J. Krismer, and A. Zell, "Markerless racket pose detection and stroke classification based on stereo vision for table tennis robots," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pp. 189–196. Accessed: Aug. 26, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8675584

[20] P. Blank, B. H. Groh, and B. M. Eskofier, "Ball speed and spin estimation in table tennis using a racket-mounted inertial sensor," in *ACM International Symposium on Wearable Computers*, Sep. 2017, pp. 2–9. doi: 10.1145/3123021.3123040.

[21] T. Gossard, J. Krismer, A. Ziegler, J. Tebbe, and A. Zell, "Table tennis ball spin estimation with an event camera," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 3347–3356. Accessed: Aug. 26, 2024. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2024W/CVsports/html/Gossard_Table_Tennis_Ball_Spin_Estimation_with_an_Event_Camera_CVPRW_2024_paper.html

[22] K. Muelling, A. Boularias, B. Mohler, B. Schölkopf, and J. Peters, "Learning strategies in table tennis using inverse reinforcement learning," *Biological cybernetics*, vol. 108, pp. 603–619, 2014.

[23] Z. Wang *et al.*, "Probabilistic movement modeling for intention inference in human–robot interaction," *The International Journal of Robotics Research*, vol. 32, no. 7, pp. 841–858, Jun. 2013, doi: 10.1177/0278364913478447.

[24] Z. Wang, A. Boularias, K. Mülling, B. Schölkopf, and J. Peters, "Anticipatory action selection for human–robot table tennis," *Artificial Intelligence*, vol. 247, pp. 399–414, 2017.

[25] K. Muelling, J. Kober, and J. Peters, “Learning table tennis with a Mixture of Motor Primitives,” in *2010 10th IEEE-RAS International Conference on Humanoid Robots*, pp. 411–416. doi: 10.1109/ICHR.2010.5686298.

[26] O. Koç, G. Maeda, and J. Peters, “Online optimal trajectory generation for robot table tennis,” *Robotics and Autonomous Systems*, vol. 105, pp. 121–137, Jul. 2018, doi: 10.1016/j.robot.2018.03.012.

[27] Y. Zhu, Y. Zhao, L. Jin, J. Wu, and R. Xiong, “Towards high level skill learning: Learn to return table tennis ball using monte-carlo based policy gradient method,” in *2018 IEEE international conference on real-time computing and robotics (RCAR)*, pp. 34–41. Accessed: Aug. 21, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8621776

[28] J. Tebbe, L. Krauch, Y. Gao, and A. Zell, “Sample-efficient reinforcement learning in robotic table tennis,” in *2021 IEEE international conference on robotics and automation (ICRA)*, pp. 4171–4178. Accessed: Aug. 21, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9560764

[29] Y. Huang, B. Schölkopf, and J. Peters, “Learning optimal striking points for a ping-pong playing robot,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 4587–4592. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7354030

[30] T. Ding *et al.*, “GoalsEye: Learning High Speed Precision Table Tennis on a Physical Robot,” Oct. 13, 2022, *arXiv*. doi: 10.48550/arXiv.2210.03662.

[31] C. Liu, Y. Hayakawa, and A. Nakashima, “Racket Control for a Table Tennis Robot to Return a Ball,” *SICE Journal of Control, Measurement, and System Integration*, vol. 6, no. 4, pp. 259–266, Jul. 2013, doi: 10.9746/jcmsi.6.259.

[32] S. W. Abeyruwan *et al.*, “i-sim2real: Reinforcement learning of robotic policies in tight human-robot interaction loops,” in *Conference on Robot Learning*, PMLR, 2023, pp. 212–224. Accessed: Aug. 26, 2024. [Online]. Available: https://proceedings.mlr.press/v205/abeyruwan23a.html

[33] M. Matsushima, T. Hashimoto, and F. Miyazaki, “Learning to the robot table tennis task-ball control & rally with a human,” in *2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme-System Security and Assurance (Cat. No. 03CH37483)*, pp. 2962–2969. Accessed: Aug. 21, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1244342/

[34] Y. Sun, R. Xiong, Q. Zhu, J. Wu, and J. Chu, “Balance motion generation for a humanoid robot playing table tennis,” in *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pp. 19–25. Accessed: Aug. 21, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6100826

[35] L. Chen, R. Paleja, and M. Gombolay, “Learning from Suboptimal Demonstration via Self-Supervised Reward Regression,” in *Proceedings of the 2020 Conference on Robot Learning*, PMLR, pp. 1262–1277. Accessed: Aug. 21, 2024. [Online]. Available: https://proceedings.mlr.press/v155/chen21b.html

[36] J. R. Harrison, “Recent problems with equipment,” *Int. J. Table Tennis Sci*, vol. 5, pp. 96–100, 2002.

[37] C. Cohen and C. Clanet, “Physics of ball sports,” *Europhysics News*, vol. 47, no. 3, pp. 13–16, 2016.

[38] T. Rémond, “Physique du rebond: application à la balle de tennis de table,” PhD Thesis, Ecole normale supérieure de lyon-ENS LYON, 2023. Accessed: Jun. 20, 2024. [Online]. Available: https://theses.hal.science/tel-04221365/

[39] D. B. D’Ambrosio *et al.*, “Robotic Table Tennis: A Case Study into a High Speed Learning System,” Sep. 06, 2023. doi: 10.15607/RSS.2023.XIX.006.

[40] “Technology to create the future. | OMRON,” A step towards OMRON’s vision of the future. | OMRON. Accessed: Oct. 17, 2024. [Online]. Available: https://www.omron.com/global/en/technology/information/forpheus/

[41] R. Clavel, “Conception d’un robot parallèle rapide à 4 degrés de liberté,” EPFL, 1991. Accessed: Oct. 01, 2024. [Online]. Available: https://infoscience.epfl.ch/record/31403/files/EPFL_TH925.pdf

[42] TableTennisDaily, “World’s Best Table Tennis Robot vs TableTennisDaily’s Dan!” Accessed: Jun. 20, 2024. [Online]. Available: https://www.youtube.com/watch?v=u3L8vGMDYD8

[43] “Achieving Human Level Competitive Robot Table Tennis.” Accessed: Oct. 17, 2024. [Online]. Available: https://sites.google.com/view/competitive-robot-table-tennis/home

[44] P. Eliseev, A. Geguev, and A. Ivanov, “Hardware Design of the Serial Link Manipulator System,” in *Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)*, IEEE, 2021, pp. 314–317. Accessed: Oct. 16, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9396660/

[45] Y. D. Patel and P. M. George, “Parallel manipulators applications—a survey,” *Modern Mechanical Engineering*, vol. 2, no. 03, pp. 57–64, 2012.

[46] O. Koç, G. Maeda, G. Neumann, and J. Peters, “Optimizing robot striking movement primitives with iterative learning control,” in *15th International Conference on Humanoid Robots (Humanoids)*, IEEE, 2015, pp. 80–87. Accessed: Oct. 17, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7363535/

[47] M. Matsushima, T. Hashimoto, M. Takeuchi, and F. Miyazaki, “A learning approach to robotic table tennis,” *IEEE Transactions on robotics*, vol. 21, no. 4, pp. 767–771, 2005.

[48] D. Casillo, “Cartesian Gantry Robots - Advantages and Applications,” Isotech, Inc. Accessed: Oct. 16, 2024. [Online]. Available: https://www.isotechinc.com/cartesian-gantry-robots/

[49] R. I. Popescu, M. Raison, G. M. Popescu, D. Saussié, and S. Achiche, “Design and development of a novel type of table tennis aerial robot player with tilting propellers,” *Mechatronics*, vol. 74, p. 102483, 2021.

[50] L. L. Cone, “Skycam-an aerial robotic camera system,” *Byte*, vol. 10, no. 10, p. 122, 1985.

[51] F. Ferlay and F. Gosselin, “A New Cable-Actuated Haptic Interface Design,” in *Haptics: Perception, Devices and Scenarios*, vol. 5024, M. Ferre, Ed., in Lecture Notes in Computer Science, vol. 5024. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 474–483. doi: 10.1007/978-3-540-69057-3_61.

[52] P. Lambert, L. Da Cruz, and C. Bergeles, “Design, modeling, and implementation of a 7-dof cable-driven haptic device with a configurable cable platform,” *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5764–5771, 2020.

[53] D. Song, X. Xiao, G. Li, L. Zhang, F. Xue, and L. Li, “Modeling and control strategy of a haptic interactive robot based on a cable-driven parallel mechanism,” *Mechanical Sciences*, vol. 14, no. 1, pp. 19–32, 2023.

[54] T. Morizono, K. Kurahashi, and S. Kawamura, “Realization of a virtual sports training system with parallel wire mechanism,” in *Proceedings of International Conference on Robotics and Automation*, IEEE, 1997, pp. 3025–3030. Accessed: Oct. 11, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/606747/

[55] G. Rosati, M. Andreolli, A. Biondi, and P. Gallina, “Performance of cable suspended robots for upper limb rehabilitation,” in *10th International Conference on Rehabilitation Robotics*, IEEE, 2007, pp. 385–392. Accessed: Oct. 11, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4428454/

[56] S. K. Banala, S. H. Kim, S. K. Agrawal, and J. P. Scholz, “Robot assisted gait training with active leg exoskeleton (ALEX),” *IEEE transactions on neural systems and rehabilitation engineering*, vol. 17, no. 1, pp. 2–8, 2008.

[57] J. D. Sanjuan *et al.*, “Cable driven exoskeleton for upper-limb rehabilitation: A design review,” *Robotics and Autonomous Systems*, vol. 126, p. 103445, 2020.

[58] Y. Mao and S. K. Agrawal, “Design of a cable-driven arm exoskeleton (CAREX) for neural rehabilitation,” *IEEE transactions on robotics*, vol. 28, no. 4, pp. 922–931, 2012.

[59] J.-P. Merlet, “Parallel robots. Vol. 74.” Springer Science & Business Media, 2001.

[60] J.-P. Merlet and C. Gosselin, “Parallel mechanisms and robots,” *Springer Handbook of Robotics*, pp. 269–285, 2008.

[61] S. Kawamura, W. Choe, S. Tanaka, and H. Kino, “Development of an ultrahigh speed robot FALCON using parallel wire drive systems,” *Journal of the Robotics Society of Japan*, vol. 15, no. 1, pp. 82–89, 1997.

[62] S. Kawamura, H. Kino, and C. Won, “High-speed manipulation by using parallel wire-driven robots,” *Robotica*, vol. 18, no. 1, pp. 13–21, 2000.

[63] J. Lamaury and M. Gouttefarde, “Control of a large redundantly actuated cable-suspended parallel robot,” in *International Conference on Robotics and Automation*, IEEE, 2013, pp. 4659–4664. Accessed: Oct. 11, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6631240/

[64] P. D. Campbell, P. L. Swaim, and C. J. Thompson, “Charlotte™ robot technology for space and terrestrial applications,” *SAE transactions*, pp. 641–648, 1995.

[65] Z. Zhang *et al.*, “State-of-the-art on theories and applications of cable-driven parallel robots,” *Front. Mech. Eng.*, vol. 17, no. 3, p. 37, Sep. 2022, doi: 10.1007/s11465-022-0693-3.

[66] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, and T.-K. Kim, “Multiple object tracking: A literature review,” *Artificial intelligence*, vol. 293, p. 103448, 2021.

[67] P. R. Kamble, A. G. Keskar, and K. M. Bhurchandi, “Ball tracking in sports: a survey,” *Artif Intell Rev*, vol. 52, no. 3, pp. 1655–1705, Oct. 2019, doi: 10.1007/s10462-017-9582-2.

[68] B. T. Naik, M. F. Hashmi, and N. D. Bokde, “A comprehensive review of computer vision in sports: Open issues, future trends and research directions,” *Applied Sciences*, vol. 12, no. 9, p. 4429, 2022.

[69] B. Cui and J.-C. Créput, “A Systematic Algorithm for Moving Object Detection with Application in Real-Time Surveillance,” *SN COMPUT. SCI.*, vol. 1, no. 2, p. 106, Mar. 2020, doi: 10.1007/s42979-020-0118-5.

[70] Z. Tu *et al.*, “A survey of variational and CNN-based optical flow techniques,” *Signal Processing: Image Communication*, vol. 72, pp. 9–24, 2019.

[71] L. Zhang and Y. Liang, “Motion human detection based on background subtraction,” in *Second International workshop on Education Technology and computer science*, IEEE, 2010, pp. 284–287. Accessed: Sep. 17, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5459067/

[72] Z. Zhong, B. Zhang, G. Lu, Y. Zhao, and Y. Xu, “An adaptive background modeling method for foreground segmentation,” *IEEE Transactions on intelligent transportation systems*, vol. 18, no. 5, pp. 1109–1121, 2016.

[73] Y. Zhang, X. Wang, and B. Qu, “Three-frame difference algorithm research based on mathematical morphology,” *Procedia Engineering*, vol. 29, pp. 2705–2709, 2012.

[74] A. Dosovitskiy *et al.*, “Flownet: Learning optical flow with convolutional networks,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2758–2766. Accessed: Sep. 17, 2024. [Online]. Available: http://openaccess.thecvf.com/content_iccv_2015/html/Dosovitskiy_FlowNet_Learning_Optical_ICCV_2015_paper.html

[75] L. Niu and N. Jiang, “A moving objects detection algorithm based on improved background subtraction,” in *eighth international conference on intelligent systems design and applications*, IEEE, 2008, pp. 604–607. Accessed: Sep. 17, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4696536/

[76] C. Zhan, X. Duan, S. Xu, Z. Song, and M. Luo, “An improved moving object detection algorithm based on frame difference and edge detection,” in *Fourth international conference on image and graphics*, IEEE, 2007, pp. 519–523. Accessed: Sep. 17, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4297140/

[77] M. Weng, G. Huang, and X. Da, “A new interframe difference algorithm for moving target detection,” in *3rd international congress on image and signal processing*, IEEE, 2010, pp. 285–289. Accessed: Sep. 18, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5648259/

[78] L. Gang, N. Shangkun, Y. Yugan, W. Guanglei, and Z. Siguo, “An improved moving objects detection algorithm,” in *International Conference on Wavelet Analysis and Pattern Recognition*, IEEE, 2013, pp. 96–102. Accessed: Sep. 18, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6599299/

[79] H. Liu, J. Dai, R. Wang, H. Zheng, and B. Zheng, “Combining background subtraction and three-frame difference to detect moving object from underwater video,” in *OCEANS 2016-Shanghai*, IEEE, 2016, pp. 1–5. Accessed: Sep. 18, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7485613/

[80] Q. Wang, J. Gao, and Y. Yuan, “Embedding structured contour and location prior in siamesed fully convolutional networks for road detection,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 230–241, 2017.

[81] P. R. Kamble, A. G. Keskar, and K. M. Bhurchandi, “A deep learning ball tracking system in soccer videos,” *Opto-Electronics Review*, vol. 27, no. 1, pp. 58–69, 2019.

[82] B. T. Naik and M. F. Hashmi, “YOLOv3-SORT: detection and tracking player/ball in soccer sport,” *Journal of Electronic Imaging*, vol. 32, no. 1, pp. 011003–011003, 2023.

[83] Md. F. A. Foysal, M. S. Islam, A. Karim, and N. Neehal, “Shot-Net: A Convolutional Neural Network for Classifying Different Cricket Shots,” in *Recent Trends in Image Processing and Pattern Recognition*, vol. 1035, K. C. Santosh and R. S. Hegadi, Eds., in Communications in Computer and Information Science, vol. 1035. , Singapore: Springer Singapore, 2019, pp. 111–120. doi: 10.1007/978-981-13-9181-1_10.

[84] M. Z. Khan, M. A. Hassan, A. Farooq, and M. U. G. Khan, “Deep cnn based data-driven recognition of cricket batting shots,” in *International Conference on Applied and Engineering Mathematics (ICAEM)*, IEEE, 2018, pp. 67–71. Accessed: Sep. 27, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8536277/

[85] K. Ma, “A real time artificial intelligent system for tennis swing classification,” in *19th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, IEEE, 2021, pp. 000021–000026. Accessed: Sep. 27, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9378695/

[86] K. Vats, M. Fani, D. A. Clausi, and J. Zelek, “Puck localization and multi-task event recognition in broadcast hockey videos,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 4567–4575. Accessed: Sep. 27, 2024. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2021W/CVSports/html/Vats_Puck_Localization_and_Multi-Task_Event_Recognition_in_Broadcast_Hockey_Videos_CVPRW_2021_paper.html

[87] J. Redmon, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016. Accessed: Sep. 27, 2024. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Redmon_You_Only_Look_CVPR_2016_paper.html

[88] J. Redmon and A. Farhadi, “YOLO9000: better, faster, stronger,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

Accessed: Sep. 27, 2024. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2017/html/Redmon_YOLO9000_Better_Faster_CVPR_2017_paper.html

[89] T. Lin, X. Zhao, and Z. Shou, "Single Shot Temporal Action Detection," in *Proceedings of the 25th ACM international conference on Multimedia*, Mountain View California USA: ACM, Oct. 2017, pp. 988–996. doi: 10.1145/3123266.3123343.

[90] T. Lin, "Focal Loss for Dense Object Detection," *arXiv preprint arXiv:1708.02002*, 2017, Accessed: Sep. 27, 2024. [Online]. Available: http://openaccess.thecvf.com/content_iccv_2017/html/Lin_Focal_Loss_for_ICCV_2017_paper.html

[91] B. Wu, F. Iandola, P. H. Jin, and K. Keutzer, "Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2017, pp. 129–137. Accessed: Sep. 27, 2024. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2017_workshops/w4/html/Wu_SqueezeDet_Unified_Small_CVPR_2017_paper.html

[92] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10781–10790. Accessed: Sep. 27, 2024. [Online]. Available: http://openaccess.thecvf.com/content_CVPR_2020/html/Tan_EfficientDet_Scalable_and_Efficient_Object_Detection_CVPR_2020_paper.html

[93] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587. Accessed: Sep. 27, 2024. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2014/html/Girshick_Rich_Feature_Hierarchies_2014_CVPR_paper.html

[94] R. Girshick, "Fast r-cnn," *arXiv preprint arXiv:1504.08083*, 2015.

[95] S. Ren, "Faster r-cnn: Towards real-time object detection with region proposal networks," *arXiv preprint arXiv:1506.01497*, 2015.

[96] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125. Accessed: Sep. 27, 2024. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2017/html/Lin_Feature_Pyramid_Networks_CVPR_2017_paper.html

[97] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969. Accessed: Sep. 27, 2024. [Online]. Available: http://openaccess.thecvf.com/content_iccv_2017/html/He_Mask_R-CNN_ICCV_2017_paper.html

[98] R. Voeikov, N. Falaleev, and R. Baikulov, “TTNet: Real-time temporal and spatial video analysis of table tennis,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 884–885. Accessed: Sep. 27, 2024. [Online]. Available: http://openaccess.thecvf.com/content_CVPRW_2020/html/w53/Voeikov_TTNet_Real-Time_Temporal_and_Spatial_Video_Analysis_of_Table_Tennis_CVPRW_2020_paper.html

[99] S. Gomez-Gonzalez, Y. Nemmour, B. Schölkopf, and J. Peters, “Reliable real-time ball tracking for robot table tennis,” *Robotics*, vol. 8, no. 4, p. 90, 2019.

[100] Y. Zhang, R. Xiong, Y. Zhao, and J. Wang, “Real-time spin estimation of ping-pong ball using its natural brand,” *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 8, pp. 2280–2290, 2015.

[101] Z. Yu *et al.*, “Design of a humanoid ping-pong player robot with redundant joints,” in *International Conference on Robotics and Biomimetics (ROBIO)*, IEEE, 2013, pp. 911–916. Accessed: Sep. 18, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6739578/

[102] Y. Zhang, W. Wei, D. Yu, and C. Zhong, “A tracking and predicting scheme for ping pong robot,” *Journal of Zhejiang University SCIENCE C*, vol. 12, no. 2, pp. 110–115, 2011.

[103] B. Toussaint and M. Raison, “Ball trajectory prediction in table tennis by combining machine learning and physical modeling,” *Applied Intelligence*, Mar. 2025.

[104] “A computational theory of human stereo vision,” *Proc. R. Soc. Lond. B.*, vol. 204, no. 1156, pp. 301–328, May 1979, doi: 10.1098/rspb.1979.0029.

[105] M. A. Fischler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981, doi: 10.1145/358669.358692.

[106] J. Tebbe, L. Klamt, Y. Gao, and A. Zell, “Spin detection in robotic table tennis,” in *IEEE international conference on robotics and automation (ICRA)*, 2020, pp. 9694–9700. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9196536/

[107] T. Gossard, J. Tebbe, A. Ziegler, and A. Zell, “SpinDOE: A ball spin estimation method for table tennis robot,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 5744–5750. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10342178

[108] S. Gomez-Gonzalez, S. Prokudin, B. Schölkopf, and J. Peters, “Real time trajectory prediction using deep conditional generative models,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 970–976, 2020.

[109] Y. Zhao, R. Xiong, and Y. Zhang, “Model based motion state estimation and trajectory prediction of spinning ball for ping-pong robots using expectation-maximization algorithm,” *Journal of Intelligent & Robotic Systems*, vol. 87, no. 3, pp. 407–423, 2017.

[110] K. Mülling, J. Kober, and J. Peters, “A biomimetic approach to robot table tennis,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 1921–

1926. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5650305/

[111] Y. Zhao, R. Xiong, and Y. Zhang, "Rebound modeling of spinning ping-pong ball based on multiple visual measurements," *IEEE Transactions on Instrumentation and Measurement*, vol. 65, no. 8, pp. 1836–1846, 2016.

[112] K. M. Kulkarni and S. Shenoy, "Table tennis stroke recognition using two-dimensional human pose estimation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 4576–4584. Accessed: Jun. 20, 2024. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2021W/CVSports/html/Kulkarni_Table_Tennis_Stroke_Recognition_Using_Two-Dimensional_Human_Pose_Estimation_CVPRW_2021_paper.html?ref=https://githubhelp.com

[113] P. Blank, J. Hoßbach, D. Schuldhaus, and B. M. Eskofier, "Sensor-based stroke detection and stroke type classification in table tennis," in *ACM International Symposium on Wearable Computers - ISWC '15*, 2015, pp. 93–100. doi: 10.1145/2802083.2802087.

[114] R. H. Bao and T. X. Yu, "Collision and rebound of ping pong balls on a rigid target," *Materials & Design*, vol. 87, pp. 278–286, 2015.

[115] R. Cross, "Impact behavior of hollow balls," *American Journal of Physics*, vol. 82, no. 3, pp. 189–195, 2014.

[116] A. Nakashima, Y. Kobayashi, Y. Ogawa, and Y. Hayakawa, "Modeling of rebound phenomenon between ball and racket rubber with spinning effect," in *ICCAS-SICE*, IEEE, 2009, pp. 2295–2300. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5335158/

[117] R. L. Andersson, "Aggressive trajectory generator for a robot ping-pong player," *IEEE Control Systems Magazine*, vol. 9, no. 2, pp. 15–21, 1989.

[118] H.-I. Lin and Y.-C. Huang, "Ball trajectory tracking and prediction for a ping-pong robot," in *9th International Conference on Information Science and Technology (ICIST)*, IEEE, 2019, pp. 222–227. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8836894/

[119] A. Van Den Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," in *International conference on machine learning*, PMLR, 2016, pp. 1747–1756. Accessed: Jun. 20, 2024. [Online]. Available: https://proceedings.mlr.press/v48/oord16.html

[120] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[121] A. Borovykh, S. Bohte, and C. W. Oosterlee, "Conditional Time Series Forecasting with Convolutional Neural Networks," Sep. 17, 2018, *arXiv*. Accessed: Jun. 20, 2024. [Online]. Available: http://arxiv.org/abs/1703.04691

[122] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, 1995, Accessed: Jul. 09, 2024. [Online]. Available:

https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=e26cc4a1c717653f323715d751c8dea7461aa105

[123] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," Dec. 11, 2014, *arXiv*. Accessed: Jul. 09, 2024. [Online]. Available: http://arxiv.org/abs/1412.3555

[124] A. Harsha, "The Ultimate Showdown: RNN vs LSTM vs GRU – Which is the Best? - Shiksha Online." Accessed: Jun. 20, 2024. [Online]. Available: https://www.shiksha.com/online-courses/articles/rnn-vs-gru-vs-lstm/

[125] R. Yu, S. Zheng, A. Anandkumar, and Y. Yue, "Long-term Forecasting using Higher Order Tensor RNNs," Aug. 23, 2019, *arXiv*. Accessed: Jun. 20, 2024. [Online]. Available: http://arxiv.org/abs/1711.00073

[126] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014, Accessed: Jul. 09, 2024. [Online]. Available: https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html

[127] P. Liu, X. Sun, Y. Han, Z. He, W. Zhang, and C. Wu, "Arrhythmia classification of LSTM autoencoder based on time series anomaly detection," *Biomedical Signal Processing and Control*, vol. 71, p. 103228, 2022.

[128] Y. Wei, J. Jang-Jaccard, W. Xu, F. Sabrina, S. Camtepe, and M. Boulic, "LSTM-autoencoder-based anomaly detection for indoor air quality time-series data," *IEEE Sensors Journal*, vol. 23, no. 4, pp. 3787–3800, 2023.

[129] X. Liu *et al.*, "Time-series forecasting based on fuzzy cognitive maps and GRU-autoencoder," *Soft Computing*, pp. 1–17, 2023.

[130] K. Merkelbach, S. Schaper, C. Diedrich, S. J. Fritsch, and A. Schuppert, "Novel architecture for gated recurrent unit autoencoder trained on time series from electronic health records enables detection of ICU patient subgroups," *Scientific reports*, vol. 13, no. 1, p. 4053, 2023.

[131] M. Abboush, C. Knieke, and A. Rausch, "GRU-based denoising autoencoder for detection and clustering of unknown single and concurrent faults during system integration testing of automotive software systems," *Sensors*, vol. 23, no. 14, p. 6606, 2023.

[132] D. Chen, H. Wang, and M. Zhong, "A short-term traffic flow prediction model based on AutoEncoder and GRU," in *12th International Conference on Advanced Computational Intelligence (ICACI)*, IEEE, 2020, pp. 550–557. Accessed: Sep. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9177506/

[133] K. Cho, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[134] N. Qadeer, J. H. Shah, M. Sharif, M. A. Khan, G. Muhammad, and Y.-D. Zhang, "Intelligent tracking of mechanically thrown objects by industrial catching robot for automated in-plant logistics 4.0," *Sensors*, vol. 22, no. 6, p. 2113, 2022.

[135] Y. Gao, J. Tebbe, and A. Zell, "A model-free approach to stroke learning for robotic table tennis," in *International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2022, pp. 1–

8. Accessed: Jun. 20, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9892776/

[136] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017, Accessed: Sep. 20, 2024. [Online]. Available: https://user.phil.hhu.de/~cwurm/wp-content/uploads/2020/01/7181-attention-is-all-you-need.pdf

[137] N. Wu, B. Green, X. Ben, and S. O'Banion, "Deep transformer models for time series forecasting: The influenza prevalence case," *arXiv preprint*, 2020, Accessed: Sep. 20, 2024. [Online]. Available: https://arxiv.org/abs/2001.08317

[138] Y. Huang, D. Büchler, O. Koç, B. Schölkopf, and J. Peters, "Jointly learning trajectory generation and hitting point prediction in robot table tennis," in *16th International Conference on Humanoid Robots (Humanoids)*, IEEE, 2016, pp. 650–655. Accessed: Sep. 23, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7803343/

[139] H. Li, H. Wu, L. Lou, K. Kühnlenz, and O. Ravn, "Ping-pong robotics with high-speed vision system," in *12th International Conference on Control Automation Robotics & Vision (ICARCV)*, IEEE, 2012, pp. 106–111. Accessed: Sep. 23, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6485142/

[140] R. Akrour, A. Abdolmaleki, H. Abdulsamad, J. Peters, and G. Neumann, "Model-free trajectory-based policy optimization with monotonic improvement," *Journal of machine learning research*, vol. 19, no. 14, pp. 1–25, 2018.

[141] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation Learning: A Survey of Learning Methods," *ACM Comput. Surv.*, vol. 50, no. 2, pp. 1–35, Mar. 2018, doi: 10.1145/3054912.

[142] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *The International Journal of Robotics Research*, vol. 40, no. 4–5, pp. 698–721, Apr. 2021, doi: 10.1177/0278364920987859.

[143] D. Ho, K. Rao, Z. Xu, E. Jang, M. Khansari, and Y. Bai, "Retinagan: An object-aware approach to sim-to-real transfer," in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 10920–10926. Accessed: Sep. 23, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9561157/

[144] OpenAI *et al.*, "Solving Rubik's Cube with a Robot Hand," Oct. 15, 2019, *arXiv*. Accessed: Sep. 23, 2024. [Online]. Available: http://arxiv.org/abs/1910.07113

[145] D. Ghosh *et al.*, "Learning to Reach Goals via Iterated Supervised Learning," Oct. 02, 2020, *arXiv*. Accessed: Sep. 23, 2024. [Online]. Available: http://arxiv.org/abs/1912.06088

[146] C. Lynch *et al.*, "Learning Latent Plans from Play," in *Proceedings of the Conference on Robot Learning*, PMLR, May 2020, pp. 1113–1132. Accessed: Sep. 23, 2024. [Online]. Available: https://proceedings.mlr.press/v100/lynch20a.html

[147] R. Singh, V. Kukshal, and V. S. Yadav, "A Review on Forward and Inverse Kinematics of Classical Serial Manipulators," in *Advances in Engineering Design*, P. K. Rakesh, A. K. Sharma, and I. Singh, Eds., in Lecture Notes in Mechanical Engineering. , Singapore: Springer Singapore, 2021, pp. 417–428. doi: 10.1007/978-981-33-4018-3_39.

[148] K. Waldron, K. Waldron, J. Schmiedeler, and J. Schmiedeler, “Handbook of Robotics Chapter 1: Kinematics,” 2007, Accessed: Oct. 16, 2024. [Online]. Available: https://faculty.cc.gatech.edu/~hic/8803-Mobile-08/slides/1_draft.pdf

[149] N. Docquier, A. Poncelet, and P. Fisette, “ROBOTRAN: a powerful symbolic gnerator of multibody models,” *Mechanical Sciences*, vol. 4, no. 1, pp. 199–219, 2013.

[150] V. R. De Angulo and C. Torras, “Learning inverse kinematics: Reduced sampling through decomposition into virtual robots,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 38, no. 6, pp. 1571–1577, 2008.

[151] A. A. Hassan, M. El-Habrouk, and S. Deghedie, “Inverse kinematics of redundant manipulators formulated as quadratic programming optimization problem solved using recurrent neural networks: A review,” *Robotica*, vol. 38, no. 8, pp. 1495–1512, 2020.

[152] A. T. Hasan, A. M. S. Hamouda, N. Ismail, and H. Al-Assadi, “An adaptive-learning algorithm to solve the inverse kinematics problem of a 6 DOF serial robot manipulator,” *Advances in Engineering Software*, vol. 37, no. 7, pp. 432–438, 2006.

[153] S. Alavandar and M. J. Nigam, “Neuro-fuzzy based approach for inverse kinematics solution of industrial robot manipulators,” *International Journal of Computers Communications & Control*, vol. 3, no. 3, pp. 224–234, 2008.

[154] M. L. Husty, M. Pfurner, and H.-P. Schröcker, “A new and efficient algorithm for the inverse kinematics of a general serial 6R manipulator,” *Mechanism and machine theory*, vol. 42, no. 1, pp. 66–81, 2007.

[155] S. Tejomurtula and S. Kak, “Inverse kinematics in robotics using neural networks,” *Information sciences*, vol. 116, no. 2–4, pp. 147–164, 1999.

[156] J. Wang, Q. Hu, and D. Jiang, “A Lagrangian network for kinematic control of redundant robot manipulators,” *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1123–1132, 1999.

[157] Y. Zhang, Z. Tan, K. Chen, Z. Yang, and X. Lv, “Repetitive motion of redundant robots planned by three kinds of recurrent neural networks and illustrated with a four-link planar manipulator’s straight-line example,” *Robotics and Autonomous Systems*, vol. 57, no. 6–7, pp. 645–651, 2009.

[158] Y. Xia and J. Wang, “A dual neural network for kinematic control of redundant robot manipulators,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 31, no. 1, pp. 147–154, 2001.

[159] E. Todorov and W. Li, “A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems,” in *Proceedings of the American Control Conference.*, IEEE, 2005, pp. 300–306. Accessed: Oct. 14, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1469949/

[160] M. Zhihong and M. Palaniswami, “Robust tracking control for rigid robotic manipulators,” *IEEE Transactions on Automatic Control*, vol. 39, no. 1, pp. 154–159, 1994.

[161] O. Koç, G. Maeda, and J. Peters, “Optimizing the execution of dynamic robot movements with learning control,” *IEEE Transactions on Robotics*, vol. 35, no. 4, pp. 909–924, 2019.

[162] S. Chen and J. T. Wen, “Neural-learning trajectory tracking control of flexible-joint robot manipulators with unknown dynamics,” in *International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 128–135. Accessed: Jun. 25, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8968608/

[163] S. Chen and J. T. Wen, “Industrial robot trajectory tracking control using multi-layer neural networks trained by iterative learning control,” *Robotics*, vol. 10, no. 1, p. 50, 2021.

[164] M. Spong, K. Khorasani, and P. Kokotovic, “An integral manifold approach to the feedback control of flexible joint robots,” *IEEE Journal on Robotics and Automation*, vol. 3, no. 4, pp. 291–300, 1987.

[165] S. Arimoto, “Learning control theory for robotic motion,” *Adaptive Control & Signal*, vol. 4, no. 6, pp. 543–564, Nov. 1990, doi: 10.1002/acs.4480040610.

[166] M. Norrlof, “An adaptive iterative learning control algorithm with experiments on an industrial robot,” *IEEE Transactions on robotics and automation*, vol. 18, no. 2, pp. 245–251, 2002.

[167] M. Meindl, D. Lehmann, and T. Seel, “Bridging reinforcement learning and iterative learning control: Autonomous motion learning for unknown, nonlinear dynamics,” *Frontiers in Robotics and AI*, vol. 9, p. 793512, 2022.

[168] S. Chen and J. T. Wen, “Adaptive neural trajectory tracking control for flexible-joint robots with online learning,” in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 2358–2364. Accessed: Jun. 25, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9197051/

[169] W. He, Z. Yan, Y. Sun, Y. Ou, and C. Sun, “Neural-learning-based control for a constrained robotic manipulator with flexible joints,” *IEEE transactions on neural networks and learning systems*, vol. 29, no. 12, pp. 5993–6003, 2018.

[170] B. Toussaint and M. Raison, “High-speed trajectory tracking on robotic arm by learning the dynamic response of the pid controller with a neural network,” 2021, Accessed: Jun. 25, 2024. [Online]. Available: https://publications.polymtl.ca/50604/

[171] M. Wang, H. Ye, and Z. Chen, “Neural Learning Control of Flexible Joint Manipulator with Predefined Tracking Performance and Application to Baxter Robot,” *Complexity*, pp. 1–14, 2017, doi: 10.1155/2017/7683785.

[172] W. Lou and X. Guo, “Adaptive Trajectory Tracking Control using Reinforcement Learning for Quadrotor,” *International Journal of Advanced Robotic Systems*, vol. 13, no. 1, p. 38, Jan. 2016, doi: 10.5772/62128.

[173] Y. Ouyang, L. Dong, Y. Wei, and C. Sun, “Neural network based tracking control for an elastic joint robot with input constraint via actor-critic design,” *Neurocomputing*, vol. 409, pp. 286–295, 2020.

[174] Y. Jiang, Y. Wang, Z. Miao, J. Na, Z. Zhao, and C. Yang, “Composite-learning-based adaptive neural control for dual-arm robots with relative motion,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 3, pp. 1010–1021, 2020.

[175] L. Kong, W. He, C. Yang, and C. Sun, “Robust neurooptimal control for a robot via adaptive dynamic programming,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 6, pp. 2584–2594, 2020.

[176] W. He, Y. Dong, and C. Sun, “Adaptive neural impedance control of a robotic manipulator with input saturation,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 3, pp. 334–344, 2015.

[177] J. H. Pérez-Cruz, I. Chairez, J. Rubio, and J. Pacheco, “Identification and control of class of non-linear systems with non-symmetric deadzone using recurrent neural networks,” *IET Control Theory & Applications*, vol. 8, no. 3, pp. 183–192, Feb. 2014, doi: 10.1049/iet-cta.2013.0248.

[178] S. J. Yoo, J. B. Park, and Y. H. Choi, “Adaptive output feedback control of flexible-joint robots using neural networks: dynamic surface design approach,” *IEEE Transactions on Neural Networks*, vol. 19, no. 10, pp. 1712–1726, 2008.

[179] H. A. Talebi, R. V. Patel, and K. Khorasani, “Inverse dynamics control of flexible-link manipulators using neural networks,” in *Proceedings. International Conference on Robotics and Automation (Cat. No. 98CH36146)*, IEEE, 1998, pp. 806–811. Accessed: Jun. 25, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/677084/

[180] R. Calandra, S. Ivaldi, M. P. Deisenroth, E. Rueckert, and J. Peters, “Learning inverse dynamics models with contacts,” in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 3186–3191. Accessed: Jun. 25, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7139638/

[181] E. Rueckert, M. Nakatenus, S. Tosatto, and J. Peters, “Learning inverse dynamics models in o (n) time with lstm networks,” in *17th International Conference on Humanoid Robotics (Humanoids)*, IEEE, 2017, pp. 811–816. Accessed: Jun. 25, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8246965/

[182] C. Yang, X. Wang, L. Cheng, and H. Ma, “Neural-learning-based telerobot control with guaranteed performance,” *IEEE transactions on cybernetics*, vol. 47, no. 10, pp. 3148–3159, 2016.

[183] T. Sun, H. Pei, Y. Pan, H. Zhou, and C. Zhang, “Neural network-based sliding mode adaptive control for robot manipulators,” *Neurocomputing*, vol. 74, no. 14–15, pp. 2377–2384, 2011.

[184] J. Trierweiler, “A systematic approach to control structure design,” 1997.

[185] Q. Li, J. Qian, Z. Zhu, X. Bao, M. K. Helwa, and A. P. Schoellig, “Deep neural networks for improved, impromptu trajectory tracking of quadrotors,” in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 5183–5189. Accessed: Jun. 25, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7989607/

[186] B. Toussaint, “LiBRTy-Lab/minimal_neural_controller: Public code of paper : Design of a minimal model-free controller for fast robotic arms trajectory.” Accessed: Jul. 16, 2024. [Online]. Available: https://github.com/LiBRTy-Lab/minimal_neural_controller

[187] J. Tebbe, Y. Gao, M. Sastre-Rienietz, and A. Zell, “A Table Tennis Robot System Using an Industrial KUKA Robot Arm,” in *Pattern Recognition*, vol. 11269, T. Brox, A. Bruhn, and

M. Fritz, Eds., in Lecture Notes in Computer Science, vol. 11269. , Cham: Springer International Publishing, 2019, pp. 33–45. doi: 10.1007/978-3-030-12939-2_3.

[188] M. N. Ahmed, M. Raison Ing. ,. Ph D., and E. Clark Erg., *Manuel technique de Unity pour la configuration de mouvements d'avatar 3D de robotique*. 2023. Accessed: Dec. 10, 2024. [Online]. Available: https://hal.science/hal-04248763

[189] M. N. Ahmed, M. Raison Ing. ,. Ph D., and E. Clark Erg., *Manuel technique de Blender pour la modélisation d'un avatar 3D de robotique*. 2023. Accessed: Dec. 10, 2024. [Online]. Available: https://hal.science/hal-04248718