Neural learning for efficient quadrotor flight control

Esteban Carvalho

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : EEATS - Electronique, Electrotechnique, Automatique, Traitement du Signal (EEATS)
Spécialité : Automatique - Productique
Unité de recherche : Grenoble Images Parole Signal Automatique

Amélioration de l'efficacité du contrôle du vol des quadrirotors par apprentissage neuronal

Neural learning for efficient quadrotor flight control

Présentée par :

Estéban CARVALHO

Direction de thèse :

Nicolas MARCHAND DIRECTEUR DE RECHERCHE, Université Grenoble Alpes	Directeur de thèse
Ahmad HABLY MAITRE DE CONFERENCES, Université Grenoble Alpes	Co-directeur de thèse
Jilles Steeve DIBANGOYE MAITRE DE CONFERENCES, INSA Lyon	Co-encadrant de thèse

Rapporteurs :

David FILLIAT
PROFESSEUR DES UNIVERSITES, ENSTA PARIS
José Fermi GUERRERO-CASTELLANOS
PROFESSEUR, Benemérita Universidad Autonoma

Thèse soutenue publiquement le **20 avril 2023**, devant le jury composé de :

Nicolas MARCHAND DIRECTEUR DE RECHERCHE, CNRS DELEGATION ALPES	Directeur de thèse
David FILLIAT PROFESSEUR DES UNIVERSITES, ENSTA PARIS	Rapporteur
José Fermi GUERRERO-CASTELLANOS PROFESSEUR, Benemérita Universidad Autonoma	Rapporteur
Denis PELLERIN PROFESSEUR DES UNIVERSITES, UNIVERSITE GRENOBLE ALPES	Président
Pedro CASTILLO-GARCIA DIRECTEUR DE RECHERCHE, CNRS DELEGATION HAUTS-DE-FRANCE	Examinateur
Guillaume ALLIBERT MAITRE DE CONFERENCES HDR, UNIVERSITE COTE D'AZUR	Examinateur

Invités :

Ahmad HABLY
MAITRE DE CONFERENCES HDR, UGA - Grenoble INP
Jilles Steeve DIBANGOYE
MAITRE DE CONFERENCES, INSA Lyon

UNIVERSITÉ DE GRENOBLE ALPES

ÉCOLE DOCTORALE SIGLE ED

Description de complète de l'école doctorale

T H È S E

pour obtenir le titre de

docteur en sciences

de l'Université de Grenoble

Mention : Automatique et Productique

Présentée et soutenue par

Estéban CARVALHO

Neural learning for efficient quadrotor flight control

Amélioration de l'efficacité du contrôle du vol des quadrirotors par apprentissage neuronal

Thèse dirigée par Nicolas MARCHAND et co-dirigée par Ahmad HABLY et Jilles Steeve DIBANGOYE

préparée au GIPSA-lab et CITI-lab

soutenue le 20 avril 2023

Jury :

Directeur :	Nicolas MARCHAND	- CNRS
Encadrants :	Ahmad HABLY	- UGA
	Jilles Steeve DIBANGOYE	- INSA Lyon
Président :	Denis PELLERIN	- UGA
Rapporteurs :	David FILLIAT	- ENSTA Paris
	José Fermi GUERRERO-CASTELLANOS	- BUAP
Examinateurs :	Pedro CASTILLO-GARCIA	- UTC
	Guillaume ALLIBERT	- Université Côte d'Azur

A toi Caleb,

mon neveu parti avant le commencement,
repose en paix.

“ *One day, I'm going to live in Theory, because in Theory everything goes perfectly...* ”

Marc Levy, *P.S. from Paris*, 2017

“ *Chacun de nous, dans sa vie, a sa propre montagne à gravir.* ”

Mike Horn, *Vouloir toucher les étoiles*, 2015

Remerciements

Je tiens à présenter ici mes remerciements à toutes les personnes qui ont participé de près ou de loin à cette épopée, qu'est le parcours de mon doctorat.

Tout d'abord, je souhaite faire un n-ième remerciement par écrit à Filliat David et Guerrero-Castellanos José Fermi d'avoir accepté la mission d'être les rapporteurs de mon manuscrit. Un grand merci également aux examinateurs, Castillo-Garcia Pedro, Allibert Guillaume et aussi au président du jury Pellerin Denis.

Je remercie mes encadrants Marchand Nicolas, Hably Ahmad et Dibangoye Jilles Steeve, d'avoir eu confiance en mes travaux tout en me laissant une grande liberté.

Merci à Mazen Alamir et Filliat David d'avoir été les membres de mon CSI et d'avoir encouragé et soutenu mon travail.

Je tiens également à remercier Gildas, « le subtil », pour la confiance placée en moi pour les enseignements à l'ENSE3 et sa disponibilité lors de mes différents questionnements.

Merci à Alina et Alain pour les enseignements proposés à Polytech et à l'IUT.

Je remercie aussi Sylvie, Christian, Patricia et bien d'autres pour nos échanges ces dernières années.

Je souhaite également vivement remercier Amaury et Jonathan, pour votre support technique et logistique. Alexandre et Matthieu, merci pour l'accueil que vous m'avez fait avant même mon arrivée au GIPSA-lab.

Merci à mes anciens enseignants que j'ai pu côtoyer au laboratoire pendant ces années de thèses: Olivier, Christophe, John, Hayatte, Nacim, Mauro, Pierre ... et les chercheurs avec qui j'ai pu échanger durant ma thèse.

Un grand merci aux RH qui ont permis de me démêler de la charge administrative que l'on adore tant en bon chercheur. Merci à toi Virginie, d'avoir pu répondre à l'ensemble de mes interrogations avec patience, lesquelles, je le reconnais, furent très nombreuses. Merci également d'avoir été à l'écoute et toujours été bienveillante. Merci Martine pour ta bonne humeur, malgré les dérangements incessants pour avoir accès à l'emploi du temps de Nicolas. Merci à toutes les autres: Akila, Maëlle, Sonia, Diane, etc.

Je remercie le service informatique pour le gain de temps apporté.

Merci à mes « stagiaires »: Pierre, Antoine K., Manon, Antoine W. et Bahauddin.

Je souhaite ici souligner le fait que cette « expérience de thèse » ne se résume pas à la rédaction de ce manuscrit, ni même à trois ans (et demi, pour mon cas) de travail. C'est une expérience de vie durant laquelle doutes, joies, chagrins, partages, désespoirs et réussites se sont vivement entremêlés. Ceci bien plus que ce à quoi l'on peut s'imaginer en début de thèse. En fait, derrière chaque manuscrit de thèse, il y a un parcours de vie, que l'on a tendance à oublier. Un cheminement inconnu au lecteur dont le manuscrit ne retrace que les réussites ou autres découvertes scientifiques. La vérité est en réalité bien au-delà de cela. Le parcours ne se déroule, bien souvent, pas comme « le jeune » chercheur l'eu imaginé. Dans mon cas, nombreux ont été les moments de doutes pendant cette période, même si j'ai bien souvent tenter de le masquer. Fort heureusement, de multiples personnes ont été présente, à mes côtés, pour me soutenir. Des personnes ouvertes au partage, à l'écoute et à l'échange qui ont permis l'achèvement de cette étape de ma vie. Je tiens à remercier tous ces « proches » qui m'ont soutenu et encouragé, que cela soit de manière amicale ou désintéressée. Les mêmes qui m'ont permis de me relever quand j'en avais le plus besoin. Ces personnes qui m'ont accepté tel que je suis et qui ont vu en moi, un collègue, un ami ou un être cher. Je désire les remercier individuellement et j'y tiens, car chacun a eu son rôle dans les rouages de ce parcours. Ce rôle, bien plus important que vous ne le pensez, m'a permis d'être là où j'en suis aujourd'hui. Enfin, cher lecteur permettez moi dans les quelques prochaines lignes, un ton plus amical et moins formel que celui qui régit la rédaction du « scientifique ».

Pierre. Tu as toujours cru en mes travaux même lorsque je doutais. Ta volonté et ta force de propositions m'ont permis d'aller au bout de cette thèse. Je suis très content d'avoir pu te rencontrer et j'espère que, tout comme moi, tu garderas de bons moments de cette collaboration.

Maxime, dit "chocho". Ami d'école et compagnon de route (on peut le dire, de galère !) de thèse. On a vécu des moments de hauts et de bas, durant cette période. On est enfin venu à bout de ce périple, contrairement aux plantes de notre bureau ! J'espère ne pas t'avoir trop fait souffert avec mes randonnées.

Chhay. D'abord collègue de bureau, de même côté de table, tu es devenu un réel ami sur qui je peux toujours compter. Ta sagesse et ta présence sont sans pareil. En écrivant ces lignes, j'entends ton rire à 120 décibels. Ne change pas.

Mariana. Ta joie, ta bonne humeur, ta capacité à rassembler font ta force. À ton départ du laboratoire, nous avons tous partagé un vide. Nous avons fait un très grand bout de parcours de thèse ensemble et je suis heureux d'avoir pu partager tous ces souvenirs avec toi.

Patrick. Merci aux moments partagés ensemble, à ton calme imperturbable et à ta bonne humeur. Même si tu es la faute de tous les problèmes de Marian, on le sait, c'est pas vrai...

À vous deux, Mariana et Patrick, ce fut un plaisir pour moi d'assister à votre mariage au Brésil. On y a passé des moments que je n'oublierai jamais. Je n'ai nul doute que notre amitié perdurera, même au-delà de la distance et de cet océan qui nous sépare.

Bob. L'homme multidisciplinaire. Partenaire de grimpe, depuis que j'ai soudainement envisagé de commencer. J'espère pouvoir continuer à progresser (et te dépasser ?). En écrivant ces lignes, je repense à nos grandes voies d'escalade en Chartreuse et en Vercors, qui resteront des épisodes inoubliables. Tu es quelqu'un de formidable, garde cette force et motivation qui t'anime.

Monica. On s'est connu à l'ENSE3, je suis content d'avoir appris à mieux te connaître suite à ton arrivée en thèse au GIPSA-lab. Ton calme, ton sourire et ta détermination font ta force. Nos périples d'escalade avec Bob resteront des histoires à compter à nos petits-enfants.

Je tiens à remercier la team intermédiaire: Agathe, Anaïs, Anna-Lena, Bob, Fanny, Loïc, Monica et bien évidemment, notre guide : l'inestimable Thierry. Mais restons concentrés, car ce n'est pas fini !

Mohamed. Lors de notre première rencontre, dans le cadre professionnel, j'étais loin de m'imaginer l'amitié qui allait naître entre nous. Ta persévérance, ta volonté, ta rigueur font de toi une personne admirable, dont on devrait prendre exemple. J'attends avec impatience le jour où tu m'apprendras la passion du surf qui t'anime. Merci pour tout ce que tu as fait pour moi.

Nicolas V. Je suis heureux d'avoir eu l'opportunité de te rencontrer mon ami. Merci pour toutes les randonnées partagées. Cela m'a permis d'aller de l'avant après une période difficile. Notre ascension de Belledonne en un jour, depuis Lancey (et non Marseille !), marque une étape formidable de nos efforts et de notre progression.

Mohamed, Nicolas, notre trio de randonnées intensives constitue un élément majeur de la réussite de ma thèse, je n'en ai nul doute. Merci à vous deux.

Andrea. Ton prénom, rime avec Picanha, coïncidence ? Je ne pense pas. A peine arrivé au laboratoire, tu as su t'intégrer et nous accueillir. Nos fous rires au Brésil, comme ailleurs, demeureront gravés dans ma mémoire. Ne change rien à ta gentillesse, ton calme et ta bonté.

Elise. Fière représentante du Langres, je te remercie pour tout ce que tu as fait pour moi. Ta présence et ta force de caractère font de toi une personne formidable, une femme forte. Extrêmement heureux d'avoir pu faire ta rencontre, je n'ai nul doute que notre amitié perdurera !

Andrea, Elise, vous formez un couple incroyable. Merci à vous deux. Il me tarde d'assister à votre union !

Houssem, dit "le grand gaillard". Je te dis : merci. Aller, salut le naze ! En réalité, j'avais envie de m'arrêter ici mais je ne peux pas, regarde ce que tu me fais écrire... Reste qui tu es. Courage, tu vas venir à bout de tous ces obstacles.

Louise. Je n'ai plus de paçoca, il va falloir que tu m'en rapportes, s'il te plaît. Je vais me préparer et un jour on pourra faire une sortie vélo longue durée avec les autres ... et contrairement à la randonnée, je serai derrière.

Lucas, dit "le papa". Maintenantt tu l'es vraiment, mais tu l'as toujours été pour la famille des doctorants. Désolé de t'avoir maltraité avec mes randonnées. Mais, en vrai, c'était sympa, non ? On n'a toujours pas fait le petit train à Grenoble, zut ! Je te souhaite du bonheur dans ta famille. Prends soin de Layana et Léticia.

Raj. Merci à toi. Je me souviendrai de ces longs échanges partagés. Il me tarde de visiter l'Inde pour en apprendre plus sur ta culture. Je te souhaite le meilleur à toi, Shivi et Aadya.

Kaouther. Merci, pour ta gentillesse et l'aide que tu as pu fournir au petit nouveau que j'étais en arrivant en Gipsa. Ta bienveillance et générosité ont fait un bien fou aux doctorants du laboratoire.

Makia. Ta soutenance restera un exemple pour moi. Tu as fait une superbe thèse ! Je te souhaite une très bonne continuation.

Maria. La vie du laboratoire, et celle en dehors, n'aurait jamais été la même sans toi. Ta force de regrouper les gens et faire la fête m'impressionnera toujours autant. Puisses-tu t'épanouir dans ton nouveau travail en Italie !

Ana. Je me souviens de nos discussions exaspérées dans ton bureau, mais au final, on s'en est sorti ! Belle rencontre de thèse, à très bientôt !

Ariel. Je dédie ces quelques mots à ta spécialité, le "Arielito" qui doit résonner dans les récits de remerciements.

Elena. Un jour j'irai en Colombie pour y découvrir ta culture... et y randonner ! Prends soin de mon avocat s'il te plait.

Mukthar. C'est mon tour, j'ai enfin fini, enfin libre, comme je le dis si souvent ! Prends soin de ta famille.

Adrien. Mon remplaçant, président du Gipsa-doc. J'espère que mes signaux de tes expériences vont t'ouvrir une grande carrière dans la recherche.

Anna. Ton sourire et ton énergie positive sont bénéfiques pour la famille des doctorants. Bonne continuation pour ta thèse !

Aurélien, l'éternel "stagiaire". À l'heure où j'écris ces lignes tu es encore en stage ! Même quand tu auras commencé ta thèse, il sera difficile de te séparer de ton surnom. Bonne continuation !

Mathias. Que dire, beaucoup trop pour tenir en quelques lignes.

Gian et Tommaso, bon courage à notre mafia italienne pour la suite de votre thèse.

Mathieu et Mathieu, vous êtes la relève. Rendez nous fiers !

Antoine, Bruce, Clément, Daniel, Florian, Hélène, Madalina, Marcello, Mohit, Quentin, Renato, Sohaïb, Suha, Tarso, mais également à mes amis de plus longues date: Ann, Annis,

Rémy, Gwendoline et j'en passe, merci !

Huy. Collègue et ami d'ENSE3 content d'avoir pu entretenir notre amitié sur ces années de thèse. À très vite pour de nombreux autres restaurants.

Laurent, dit "lolo". Notre escapade à Milan et ses alentours, organisée d'une main de maître, fut une vrai bouffée d'énergie. Garde ta positivité et ta bienvaillance, à très bientôt !

Jean-Philippe, dit "jipouille". On a passé un super séjour à Aix-les-Bains. Fais moi signe quand tu voudras faire la traversée de Chartreuse, ton guide est prêt !

Julien. Merci pour les moments partagés. A très bientôt dans les montagnes ! Préviens moi, dès la sortie de tes romans, il me tarde de les lire.

Bruno. Note : mettre la musique "Roule". Merci à tous ces appels mensuels, à m'écouter parler pendant des heures. À très bientôt mon ami !

Katia. Il est des rencontres qui changent une vie, tu en fais partie. Les moments passés ensemble m'ont beaucoup appris. Tu es une femme forte, un jour tu t'en rendras compte.

Rémi. Rémibro. Mon frère de cœur. Notre amitié m'est inestimable. Je te remercie pour tous les moments partagés ensemble et j'attends déjà avec impatience nos prochaines aventures !

Bien évidemment, je ne serai pas là sans ma famille : Maman, Papa, Maxime, Anaïs, Marina, Abel. Vous représentez tout pour moi. Mais également à tous les autres, Raphaëla, Julien, Quentin, Ronan, Quentin, Axel, Bérénice, Claire-Lise, mes cousins, cousines, oncles, tantes, etc. Je le sais, vous serez toujours là pour moi, même dans les moments les plus difficiles. Je remercie la bénédiction que l'on m'a accordé d'avoir une famille aimante et bienveillante. Je n'ai en fait qu'une chose à vous dire : je vous aime.

A toi Maxime, mon frère, mon jumeau, ma moitié. Je dédie ces dernières lignes. Comme je me prête à le dire si souvent : ceux qui ne te connaissent pas, ne me connaissent pas. Sans toi, je ne serai rien. Je t'aime mon frère.

Contents

List of Figures

List of Tables

Table of abbreviations and acronyms

AI	Artificial Intelligence
DFT	Discrete Fourier Transform
DNN	Deep Neural Network
EKF	Extended Kalman Filter
ESC	Electronic Speed Controller
IMU	Inertial Measurement Unit
LQR	Linear-Quadratic Regulator
MOCA	Motion Capture
MPC	Model Predictive Control
MSE	Mean Squared Error
NADAM	Nesterov-Accelerated Adaptive Moment (algorithm)
NN	Neural Network
PID	Proportional Integral Derivative (regulator)
PWM	Pulse Width Modulation
RC	Remote Control
RL	Reinforcement Learning
RMSE	Root Mean Squared Error
ROS	Robot Operating System
SGD	Stochastic Gradient Descent
UAV	Unmanned Aerial Vehicle
VRPN	Virtual Reality Peripheral Network

Introduction

General introduction

Unmanned Aerial Vehicles (UAVs), and in particular quadrotors, have been experiencing a growing interest during last decades. That exponential interest can be explained in several ways, in particular, thanks to their small size, their ease of construction, low maintenance and repair cost. The quadrotor high maneuverability, ability to hover, to perform indoor navigation or acrobatic flights have made it possible to have a wide range of usages. Thus, many applications have emerged and have continuously developing since. To introduce some applications of this disruptive technology, we can cite: industrial surveillance [Sil+21], infrastructure inspections [Gu+20], safety and rescue missions [Bir+11], cinematography [TG+17], merchandise transport [Sch20] or aerial manipulation [KCK13]. The figure n°1 illustrates some of possible usage of quadrotors. More uses can be found in the survey [Sha+19]. These numerous applications reveal a vast field of research activity. Indeed, it requires the design of reactive piloting algorithms, precise vision algorithms or high-performance obstacle detection technologies, for instance. It also concerns the work on the autonomy of the batteries or the performance of the different components that compose it.

Although many research and development efforts are currently focused on perception tools, as well as their integration into trajectory generation, or the collaboration of several aircraft in flight (e.g. swarm flight), the control application for the trajectory tracking performance still raises great interest in the research field. A lot of different control techniques have been developed and tested on quadrotors: Proportional Integral Derivative (PID) [Sal+10], backstepping control [BS], Model Predictive Control (MPC) [BM14], adaptive control [DAL13], bounded control [GC+11] or event-based control [Dur+18]. However, quadrotors are highly nonlinear systems and their increasingly varied uses subject them to conditions, environments and disturbances that are not often considered in the controller design. For instance, the presence of wind gusts in the context of surveillance missions or the presence of an additional suspended payload when transporting goods. All these effects must be taken into account and are sometimes difficult to integrate and overcome using standard control methods.

With the tremendous advances in the field of Artificial Intelligence (AI), machine and deep learning technologies have progressively moved towards quadrotors. Thus, many data-driven methods have emerged. Promising results of AI in perception field and in a lot of other domains, motivated the use of such approaches to improve quadrotors flights. Furthermore,

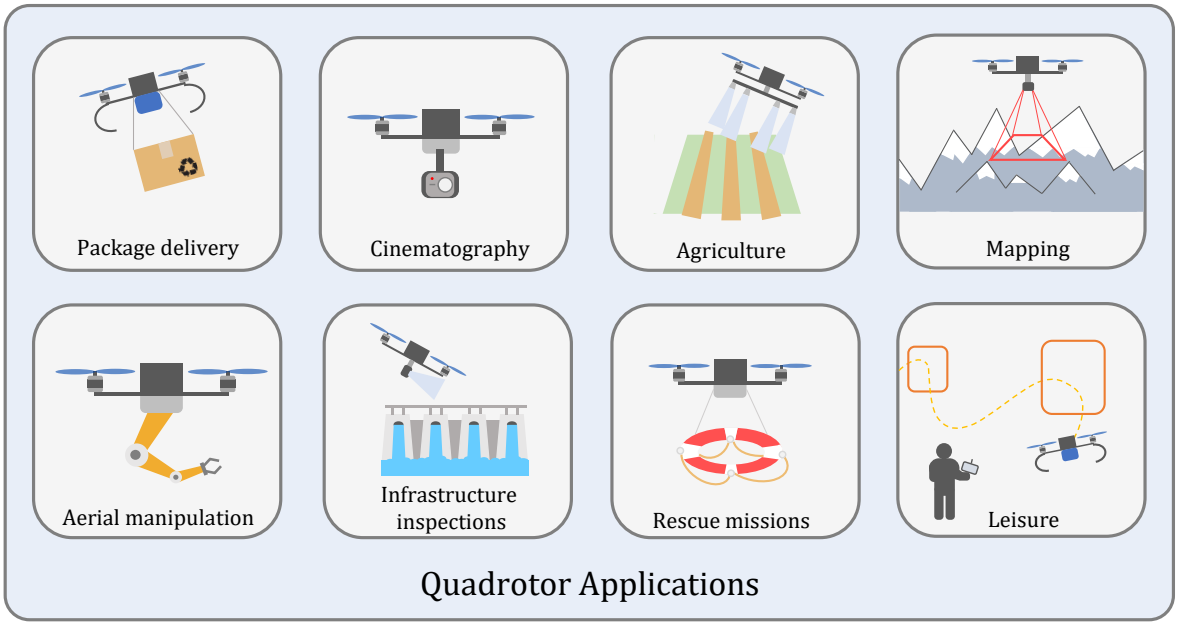


Figure 1: Illustration of quadrotor drone usage.

the use of such intelligent algorithms is encouraged by the ease of learning thanks to the large amount of data generated during flights, coming from multiple sensor sources. AI has thus constituted a research axis for possible improvement of control algorithms for trajectory tracking. For example, Neural Network (NN) have been integrated into the PID tuning process [Wan+15]; [Efe11]. New solutions propose to integrate Deep Neural Network (DNN) in MPC schemes [Kau+19]; [Shi+19]. Reinforcement Learning (RL) methods have also been developed and applied to quadrotor control [Hwa+17]; [Lam+19]; [Zha+22]. Numerous data-driven methods are thus emerging to enhance quadrotor flight performance.

However, the development of such adaptive and learning-based algorithms has led to an increase of complexity of the proposed algorithms. The use of such methods requires a rather long and tedious tuning process in order to obtain satisfactory performance in mission situations. In addition, such algorithms require a large amount of data to learn a controller. In the case of RL algorithms, it will require a large number of trials and errors. This is particularly problematic for quadrotor systems which are highly nonlinear systems. The many potential crashes before learning a performing controller is not directly feasible experimentally. This requires a lot of simulation beforehand and a transfer of the learned controller into the real world, which takes a lot of time and effort compared to the design of a more standard controller. Thus, the use of such methods is not a miracle solution and still requires research efforts to improve these drawbacks and propose algorithms that are both simple and efficient.

Thesis scope

The work presented in this document lies within the trendy research area of the last decade, which aims at merging AI and standard control techniques in order to improve the flight performance of auto-piloted quadrotors.

By flight performance improvements, one means to perform accurate trajectory tracking with reduced overshoots and limited oscillations. The trajectory tracking is used here for scenarios typically faced during a mission situation of the quadrotor. It is requiring a reactive control, robust to the wind, being able to manage payloads or to fly close to the ground. This includes having an autopilot robust to the various existing disturbances, both internal (control offsets, model errors, additional load, etc.) and external (ground effect, wind gusts, unmodeled effects, etc.). As stated earlier, many techniques to achieve this goal have been developed. But their major disadvantage is that, very often, presented algorithms are very complex and demanding in terms of data and tuning process. Furthermore, techniques relying purely on AI are not reliable from a safety point of view for the system.

The aim of this work is to be able to easily and quickly automatically drive a quadrotor, with a very limited initial knowledge on it, using a mix between standard control approaches and learning methods. This is achieved by an intelligent control able to correct its errors made during flights. Here, we wish to easily setup an initial controller, on which the user specifies his performance requirements. Afterwards, the controller self-improves using artificial intelligence techniques. Thus, a quadrotor with a given initial control architecture, possibly unknown, could be improved with a learning module that would allow to optimize the flight performance while guaranteeing the non-destabilization of the closed-loop system by the learning-based controller. The latter will allow to model effects not taken into account in the initial controller, because they are often very complex (e.g.: blade flapping effect). It will overcome the approximations due to the linearizations possibly done (e.g.: linear control). Also, it will learn unknown effects visible in the measurements (e.g.: wind), as well as the uncertainties related to the modeling (e.g.: bad estimation of the parameters), etc. Ultimately, the AI-based controller will be able to handle all these errors, improving tracking and bringing the quadrotor behavior closer to the initially desired performance.

The framework of this thesis involves the study and development of quadrotor control algorithms. Thus, we will pay a particular attention on their implementation and on the experimental tests carried out within the GIPSA-lab aerial platform. Numerous efforts have been made for the success of the proposed algorithms in practice. We have determined that the experimental part, including the technical details, will be included in this manuscript and given equal weight to the theoretical aspects.

Structure of the manuscript

The manuscript is organized as follows:

- **Quadrotor modeling**: The first chapter introduces the quadrotor dynamics commonly used in literature. It begins with a brief introduction of the quadrotor functioning. Then, necessary frames and notations used to model the device are presented. An extensive model used for simulations and for the controller validation is given using Newton-Euler formalism, by developing a balance of forces and torques that are applied on the system. A simplified model is ultimately outlined, this model is the commonly presented model in literature.

- **Experimental setup**: The chapter two focuses on the experimental setup used to test and validate proposed controllers. It presents in details the quadrotor used and its hardware architecture. The experimental room is also presented. A description of ROS mechanisms and how data are communicated and collected is done afterward. The controller architecture experimentally implemented is then explained. Deeper explanations are provided on PX4 software to perform low-level control. Finally, data processing for learning purposes is explained.

- **Integration of learning into control for linear behavior fitting**: The chapter three presents the first proposed approach for quadrotor flight tracking improvement using artificial intelligence. A Deep Neural Network is trained using flight data in order to learn unknown dynamics and is introduced in the control loop to cancel undesired behavior. It is based on a standard cascaded controller utilizing linear tools to control the different loops: position, velocity and attitude. The proposed solution is a several step procedure: first performing flights with the original controller and learning the errors made during flight. The following step is the improved flight using the proposed DNN-based controller. Results are presented on both simulations and experimental tracking scenarios.

- **Towards an online implementation: event-based neural learning**: The chapter four is about enhancing the firstly proposed solution, presented in the previous chapter. It focuses on improving main drawbacks of the previous solution: the offline learning and the limited stability guarantees. This newly proposed approach is an event-based learning strategy. It learns or re-learns a DNN and corrects quadrotor tracking in an iterative way. The strategy is based on two main events linked to stability and tracking performance. The solution is tested in simulations and real experiments, to assess its effectiveness. It mainly corrects internal model errors but also external disturbances, such as wind gusts perturbations.

- **Neural controller for unknown payload transportation**: The chapter six is about designing a controller for unknown suspended payload transportation. The task under consideration is more difficult than standard missions as it adds a disturbance acting like a pendulum, requiring a more efficient initial controller. The proposed solution

is mix between nonlinear linearizing control and a windowed deep neural network for estimating and correcting the payload dynamics. The proposed algorithm is directly tested via experimental transportation scenarios. It demonstrates the effectiveness of the approach in comparison to solutions including high-gain observers.

- Further details on disturbance observers are given in appendix **Disturbances observers** and a general overview of Deep Learning is given in appendix **Deep Learning generalities**.

Publications

This thesis led to the publication and submission of the following papers:

[Car+22] Esteban Carvalho, Pierre Susbielle, Ahmad Hably, Jilles Dibangoye, Nicolas Marchand. "Neural Enhanced Control for Quadrotor Linear Behavior Fitting." In: *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, June 2022.

[Submitted] Esteban Carvalho, Pierre Susbielle, Ahmad Hably, Jilles Dibangoye, Nicolas Marchand. "Neural Network Feedback Linearization Control for Quadrotor Flight with Unknown Suspended PayLoad."

[Submitted] Esteban Carvalho, Pierre Susbielle, Ahmad Hably, Jilles Dibangoye, Nicolas Marchand. "Event-based Neural Learning for Quadrotor Control."

These articles are the objects of the different chapters presented in this document.

Open-source software

A part of the produced code, including programs to perform experimental tests, is made available on Github. It is freely accessible via the following link: `https://github.com/gipsa-lab-uav/uav_control_ai`

Funding Sources

This work has been partially supported by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01) and ROBOTEX 2.0 (Grants ROBOTEX ANR-10-EQPX-44-01 and TIRREX ANR-21-ESRE-0015) funded by the French program Investissements d'Avenir.

Chapter I

Quadrotor modeling

This chapter introduces the quadrotor aerial device used in this thesis. After a brief description of the functioning of the aircraft, the formalism used to describe spatial movement of the vehicle is given. Then, an exhaustive description of the model is described. This first model includes several known possible aerodynamic effects that operate on the quadrotor. A simplified model, more commonly used in the control community, is finally derived from the previous one.

Contents

I.1 Quadrotor movement

We introduce here basic elements of quadrotor design and displacement, before describing its complete model. More technical details, about the quadrotor used for experimental purposes, its electronics, its micro-controllers, etc. will be given in the following dedicated chapter II.

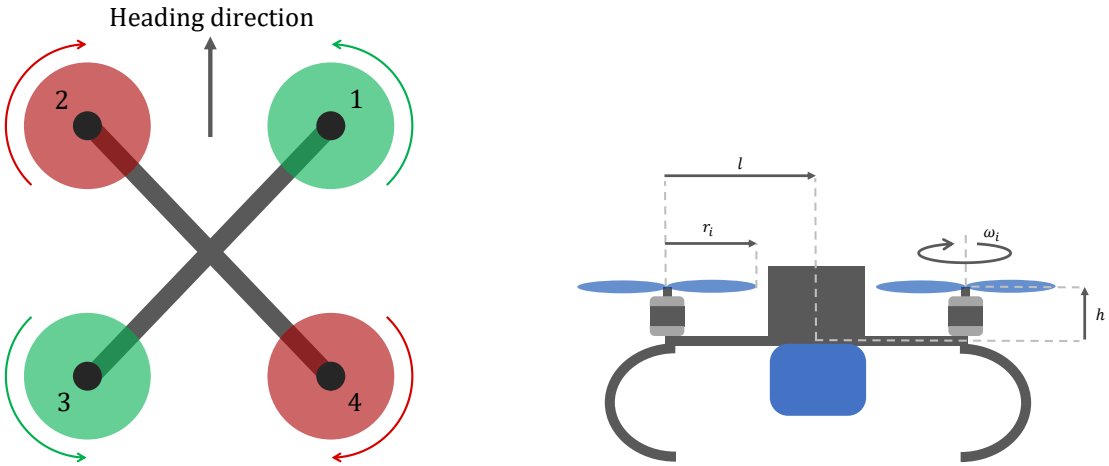
A quadrotor is an unmanned aerial vehicle composed of four motors attached to a rigid cross-shaped frame. The frame allows to embed electronics: a micro-controller, four motors and a battery. The propellers {1 to 4} are mounted such as {1} and {3} are rotating anti-clockwise and {2} and {4} are rotating clockwise, see Fig. I.1a for numbering disposition. Driving the engines in the opposite direction cancels the induced torque of each motors to turn the propellers. By piloting each motor individually and in a coordinated manner, desired motion in the 3D space can be achieved: translations, rotations and combinations between both.

In the following, we will consider an orthogonal frame defined by its vectors units $\{\vec{x}, \vec{y}, \vec{z}\}$.

For quadrotors, there exist two configurations for the heading direction:

- $X4$-*configuration*: the heading direction, along $\vec{x}$, is done between motors {1} & {2} and axis $\vec{y}$ is between {2} & {3}.
- $+4$-*configuration*: the heading direction, along $\vec{x}$, is done in the direction of motor {1} and axis $\vec{y}$ is along motor {2}.

In the remaining of the manuscript, we will consider the most used configuration, the $X4$-configuration, as represented in Fig. I.1a.



(a) $X4$-configuration (b) Quadrotor side view

Figure I.1: Quadrotor general description.

A quadrotor is an under-actuated system, it has 6 degrees of freedom, spatial and angular positions, but only 4 actuators, the motors. However, by controlling the motors jointly one can achieve motion in space. Possible motions are described in Fig. I.2. In this figure, a lighter color represents a lower motor speed and reversely a darker color stands for a faster motor speed.

We first described rotation motions, as they are at the basis of all quadrotor movements. Rotations are performed by a differential of angular speed between two pairs of motors:

- For *roll* motion: between pairs of motors {1, 4} and motors {2, 3}, see Figs. I.2 - a,
- For *pitch* motion: between pairs of motors {1, 2} and motors {3, 4}, see Figs. I.2 - b,
- For *yaw* motion: between pairs of motors {1, 3} and motors {2, 4}, see Figs. I.2 - c.

The hovering flight is achieved when the total thrust force generated by all motors is compensating the gravitational force. Translation vertically is done by applying more or less thrust than the gravitational force: increasing or decreasing all angular speed in a simultaneous way allows the quadrotor to go upward or downward, see Fig. I.2 - d.
Finally, horizontal motions (along $\vec{x}$ and $\vec{y}$) is accomplished by making first a rotation by *pitching* or *rolling* and using a total thrust for maintaining it in the horizontal plane.

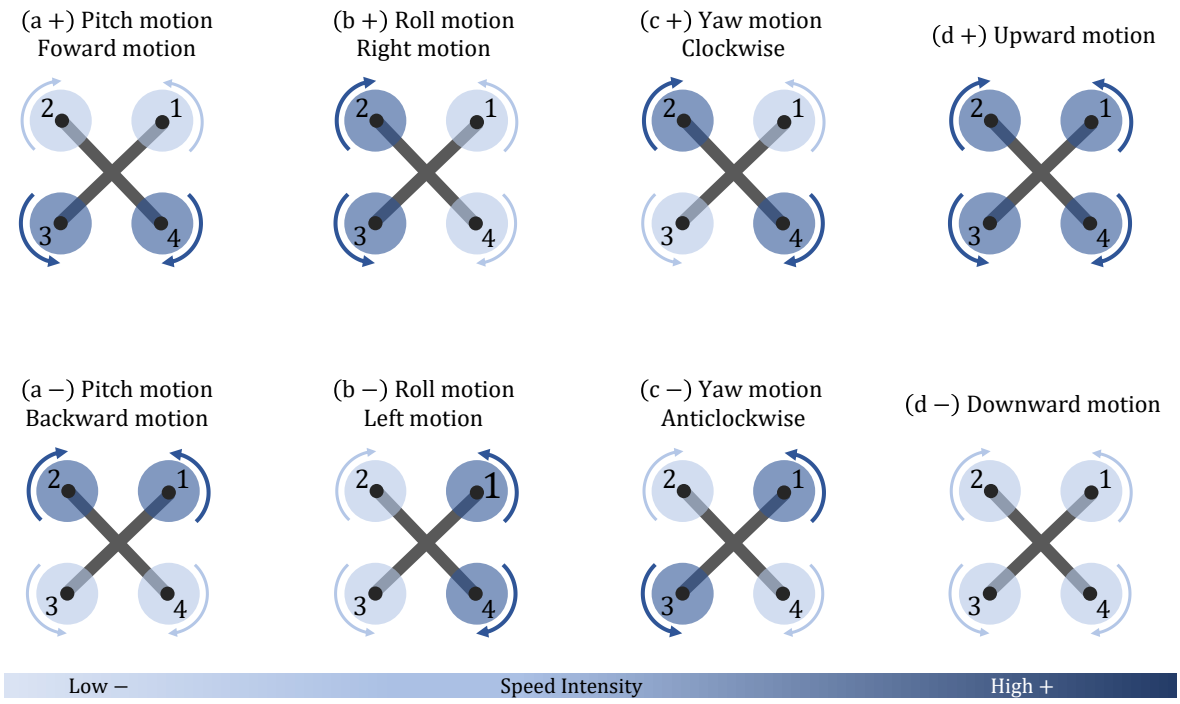


Figure I.2: X4 - Quadrotor motion description.

I.2 Frames and formalism

We define here the formalism of frames used to describe the motion of the rigid body. It will be used for all the remaining of the manuscript. Let $\{\vec{x}, \vec{y}, \vec{z}\}$ be the unit vectors of each axis without linking them to a reference frame. Two important frames must be defined:

- The *inertial frame*: let $\{\mathcal{I}\}$ be the right-hand orthonormal inertial frame described by its units vectors $\{\vec{i_1}, \vec{i_2}, \vec{i_3}\}$. Thus in $\{\mathcal{I}\}$ one has $\vec{i_1} = \vec{x}$, $\vec{i_2} = \vec{y}$ and $\vec{i_3} = \vec{z}$.
- The *body-fixed frame*: let $\{\mathcal{B}\}$ be the right-hand frame attached to the rigid body, described by its units vectors $\{\vec{b_1}, \vec{b_2}, \vec{b_3}\}$. In $\{\mathcal{B}\}$ one has $\vec{b_1} = \vec{x}$, $\vec{b_2} = \vec{y}$ and $\vec{b_3} = \vec{z}$.

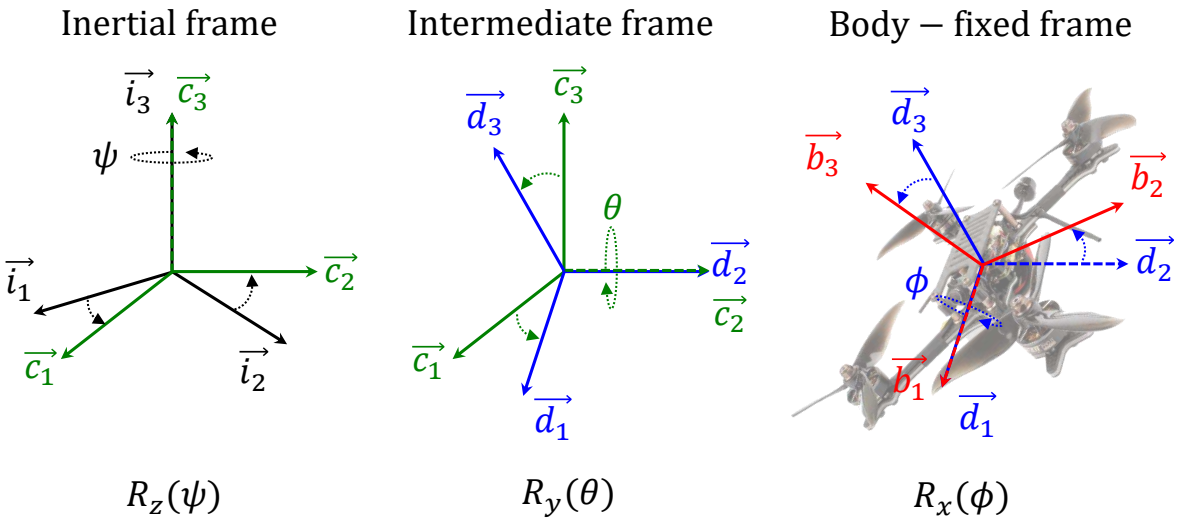


Figure I.3: Illustration of the frames representation using Tait-Bryan angles. The inertial frame on the left, intermediate frames in the middle and the body fixed frame, on the right.

The orientation of rigid-body frame $\{B\}$ in the inertial frame $\{\mathcal{I}\}$ can be defined by the rotation matrix $\boldsymbol{R} \in \mathbf{R}^{3\times3}$ and belongs to special orthogonal group $SO(3)$. It can then be written that: $\vec{b_1} = \boldsymbol{R}\vec{i_1}$, $\vec{b_2} = \boldsymbol{R}\vec{i_2}$ and $\vec{b_3} = \boldsymbol{R}\vec{i_3}$. To model the rotation from $\{\mathcal{I}\}$ to $\{\mathcal{B}\}$ one uses Euler formalism, using Tait-Bryan angles [GPS02]. The intrinsic Z-Y-X Euler formalism is defined as follows:

- A first rotation is performed around z axis of the $\{\mathcal{I}\}$ frame. This rotation is defined by the *yaw* angle ψ. A new intermediate frame is obtained.
- Secondly, a rotation of *pitch*, θ, around $\vec{y}$ axis of this intermediate frame is done, a second intermediate frame is defined.
- Finally, the new frame is rotated thought *roll* angle, ϕ, about the $\vec{x}$ axis obtained.

It results in the body-fixed frame $\{\mathcal{B}\}$. One can then define $\boldsymbol{R}$ matrix using Euler angles:

$$\begin{aligned}\boldsymbol{R} &= \boldsymbol{R}_z(\psi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_x(\phi),\\ &= \begin{bmatrix} c_\psi & -s_\psi & 0\\ s_\psi & c_\psi & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix} c_\theta & 0 & s_\theta\\ 0 & 1 & 0\\ -s_\theta & 0 & c_\theta\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & c_\phi & -s_\phi\\ 0 & s_\phi & c_\phi\end{bmatrix},\\ \boldsymbol{R} &= \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi\\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi\\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta\end{bmatrix},\end{aligned} \tag{I.1}$$

where cosines and sines are defined using following abbreviations: $c_\cdot := \cos(\cdot)$ and $s_\cdot := \sin(\cdot)$.

A relation between the time derivatives of the Euler angles and the angular velocities $\Omega = [p\ , q\ , r]^T$ can be drawn:

$$\Omega = \begin{bmatrix}\dot\phi\\ 0\\ 0\end{bmatrix} + \boldsymbol{R}_z\Omega\begin{bmatrix}0\\ \dot\theta\\ 0\end{bmatrix} + \boldsymbol{R}_z\boldsymbol{R}_y\begin{bmatrix}0\\ 0\\ \dot\psi\end{bmatrix}.$$

The angular speed vector Ω can be expressed using Euler angles derivatives using the following relation:

$$\Omega = \boldsymbol{W}^{-1}\begin{bmatrix}\dot\phi\\ \dot\theta\\ \dot\psi\end{bmatrix}, \tag{I.2}$$

where matrix $\boldsymbol{W} \in \mathbf{R}^{3\times3}$ is called the Wronskian matrix. For *Z-Y-X* Euler convention it is expressed as follows:

$$\boldsymbol{W} = \boldsymbol{W}(\phi,\theta,\psi) = \begin{bmatrix}1 & s_\phi t_\theta & c_\phi t_\theta\\ 0 & c_\phi & -s_\phi\\ 0 & s_\phi/c_\theta & c_\phi/c_\theta\end{bmatrix}, \tag{I.3}$$

where tangent function is denoted as: $t_\cdot = \tan(\cdot)$.

It can be noted that the Wronskian matrix is singular for $\theta = \pi/2 + k\pi$ for $k \in \mathbb{Z}$. In fact, two of the three axes are identical in that configuration. This issue is also known as *gimbal lock*. To solve this problem other representations exist, in particular quaternions representation. The quaternion formalism is not developed in this thesis, as the trajectory generation used will not require acrobatic motions implying that condition. More details about quaternions formalism can be found in the following technical report [BHD13] for instance.

I.3 Actuator model description

In this section, we present the model of the actuators mounted on a quadrotor: four three-phases brushless motors linked to propellers. A brushless motor is a synchronous motor and it is composed of a stator with electromagnets and a rotor built with permanent magnets. Since it does not have commutators and brushes, it is needed to switch current in stator coils to create a driving torque. In order to achieve it, commutation is made by an electronic unit.

I.3.1 Brushless motor model

The model presented here is an equivalent mono-phase model of the three-phases device, it is represented in Fig. I.4. We associate to the motor $i \in \{1, 2, 3, 4\}$ its rotation speed ω_i and its voltage supply as u_i.

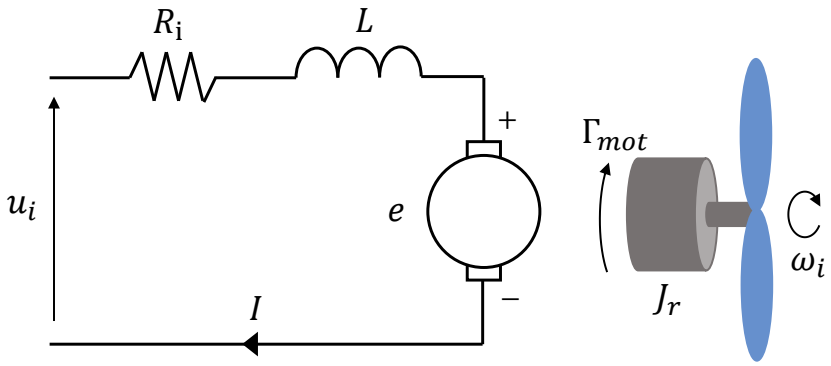


Figure I.4: Brushless motor equivalent mono-phase circuit.

The **electrical equation** (I.4) can be obtained:

$$L\frac{d\omega_i}{dt} = u_i - R_i I - e(t), \tag{I.4}$$

where following notations are used:

- $e(t) = K_e\omega_i[V]$: counter-electromotive force or back emf,
- $I[A]$: armature current,
- $K_e[V/(rad/s)]$: back emf constant or motor velocity constant, given by the manufacturer or estimated via a no-load test,
- $R_i[\Omega]$, the internal resistance of a motor phase, given by the manufacturer or measured at the terminals of a motor phase,
- $L[H]$: the internal inductance of a motor phase, given by the manufacturer or measured.

Concerning the **mechanical equation** (I.5), we have:

$$J_r \frac{d\omega_i}{dt} = \Gamma_{mot} - \Gamma_{load}, \tag{I.5}$$

where following notations are used:

- $\Gamma_{mot} = K_m I[Nm]$: motor torque,
- $K_m[Nm/A]$: torque constant,
- $J_r[kgm^2]$: inertia of rotating parts reduced to the motor axis: shaft, propeller, screw,
- $\Gamma_{load}[Nm]$: the load torque related to the motor axis. Here, the friction torque of the blades in the air is given by $\Gamma_{load} = c_D \omega_i |\omega_i|$,
- $u_i[V]$: voltage at the terminals of a motor phase limited by a maximum saturating value. This maximum voltage corresponds to the maximum rotation speed.

As motors used are small, we can consider that the inductance L is very small. Also by assuming that $K_m = K_e$, the second order DC motor dynamics are approximated by the first order equation given by eq. (I.6):

$$\dot{\omega}_i = -\frac{K_m^2}{J_r R_i}\omega_i - \frac{1}{J_r}\Gamma_{load} + \frac{K_m}{J_r R_i}\,\mathrm{sat}_{u_i}(u_i). \tag{I.6}$$

I.3.2 Forces generated by propellers

When the propeller is being driven by the motor, it generates a force perpendicular to its plane of rotation, in the ideal case: with no apparent wind nor other disturbance. The steady-state thrust generated by the *i-th* motor F_i along $\vec{b}_3$, [Lei06] is given by:

$$F_i = T_i \vec{b}_3 = C_T \rho A_{ri} r_i^2 \omega_i^2 \vec{b}_3, \tag{I.7}$$

with following variables:

- $C_T[-]$: thrust coefficient,
- $\rho[kg.m^{-3}]$: air density,
- $A_{ri}[m^2]$: motor disk area,
- $r_i[m]$: radius of the motor,
- $\omega_i[rad.s^{-1}]$: angular speed of i.

In practice, we use c_T coefficient such that:

$$T_i = c_T \omega_i^2. \tag{I.8}$$

This coefficient is a constant estimated from a static thrust test. The total thrust (I.9) applied to the rigid-body is given by the sum of each motors individual thrust force term (I.8):

$$T = \sum_{i=1}^{N=4} |T_i|. \tag{I.9}$$

The thrust force vector is then expressed by the total of individual thrust forces $F_{T_i} = T_i \vec{b_3}$:

$$F_T = \sum_{i=1}^{N=4} |T_i| \vec{b_3} = T \vec{b_3}. \tag{I.10}$$

Illustration of the four forces generated and total thrust force is given in Fig. I.5.

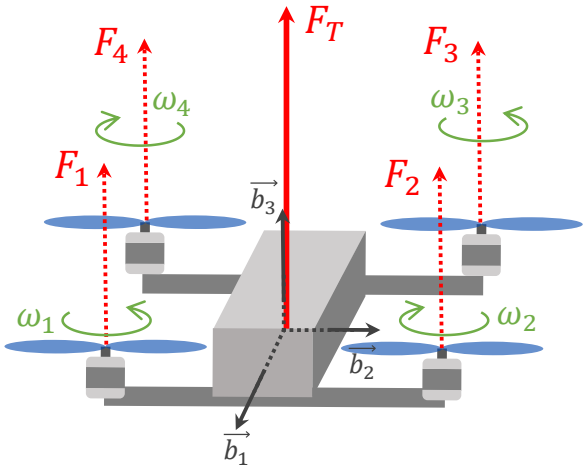


Figure I.5: Illustration of the forces generated by the four propellers.

In next section, this thrust force vector will be detailed, considering flapping effect.

I.4 Quadrotor complete model

In this section, we present a comprehensive model of a general quadrotor given in the $X4$-configuration. It will include main known effects that are well described in the literature, [Man15]; [MKC12]; [Bou07], etc. This model will be referred as *complete model* in the following of this document. The complete modeling is obtained by making the balance of forces and torques that are applied on the device.

The quadrotor center-of-mass position is denoted by $\xi := [x\ , y\ , z]^T \in \{\mathcal{I}\}$. Its linear velocity, given in $\{\mathcal{I}\}$, is $v := [v_x\ , v_y\ , v_z]^T$. The Euler angles vector is $\zeta := [\phi\ , \theta\ , \psi]^T$,

whose components correspond to roll, pitch, and yaw angles, respectively. $\Omega := [p\ , q\ , r]^T$ denotes the angular velocity vector expressed in the body frame $\{\mathcal{B}\}$. Let m be the total mass of the quadrotor, including battery, and J be the inertia matrix expressed in the body fixed frame $\{\mathcal{B}\}$.

I.4.1 Balance of forces

In this subsection, we enumerate main known forces that are applied to the rigid-body, in order to next give the model of quadrotor:

- Thrust force F_T: It is the total thrust force vector induced by the sum of all forces generated by propellers F_i (I.7), F_T was given in previous section by eq. (I.10)

- Drag force F_v: It is the resistance force of the air across the surface of the quadrotor. We can define it along each axis of the quadrotor $i \in \{x, y, z\}$, according to [Man15]:

$$F_v^i = -\frac{1}{2}\rho C_i A_{f_i} |v_{i|_\mathcal{B}}| v_{i|_\mathcal{B}}, \tag{I.11}$$

  with:

  - $\rho[kg.m^{-3}]$: air density,
  - $C_i[m.^{-1}]$: drag coefficient,
  - $A_{f_i}[m^2]$: fuselage area, projected on the plane orthogonal to the i-axis,
  - $v_{i|_\mathcal{B}}[m.s^{-1}]$: velocity expressed in $\{\mathcal{B}\}$ along axis i.

  Defining $\boldsymbol{K_v} = \frac{1}{2}\rho C_i A_{f_i} \boldsymbol{R}$, we can express the drag force vector, using (I.11) and v given in $\{\mathcal{I}\}$:

$$F_v = -\boldsymbol{K_v}|v|v. \tag{I.12}$$

- Weight force F_p: It is the force generated by gravity. Using g the free-fall acceleration, it is expressed as (I.13):

$$F_p = -mg\vec{i_3}. \tag{I.13}$$

- External disturbance force F_{ext}: It is an additional force term that includes external forces, as wind force for instance, but it also includes non-modeled and unknown effects. It is expressed in inertial frame.

There exist several external disturbances that affects the dynamic of the quadrotor and particularly the thrust force: the ground effect and the blade flapping. We aim now at explaining and describing the impact of these two effects on the system.

Ground effect: At low altitude, near ground surface, an additional effect is added to the whole dynamic: the ground effect (see Fig. I.6). It is the result of a local increase in air density due to the presence of the ground. Thrust of a horizontal propeller is thus accentuated with the proximity of the ground. It greatly affects quadrotor behavior. The ground effect is generally avoided by flying at high altitude but can be a real issue when the quadrotor must get closer to the ground.

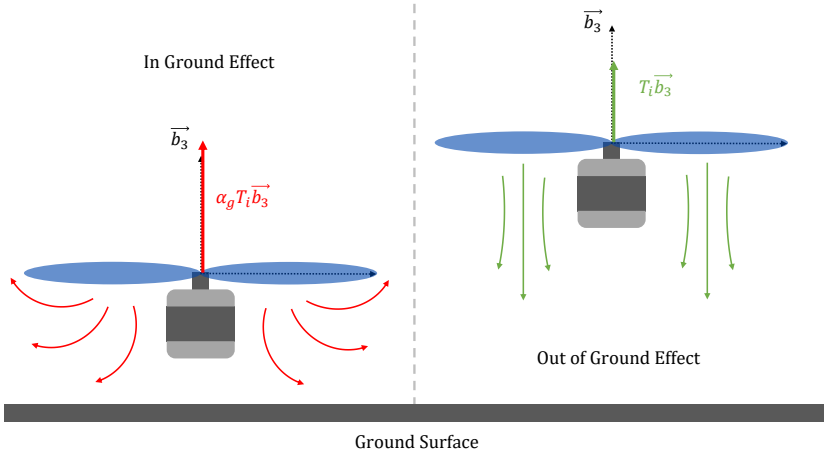


Figure I.6: Illustration of the ground effect.

The ground effect is modeled here according to the work of [Che+55] and [Dan+15] by:

$$\frac{T_{in}}{T_{out}} = 1 - \rho_g \left(\frac{r_i}{z+h} \right)^2,$$

where $T_{in} = T$ is the input thrust, T_{out} is the true thrust generated by the motors and with:

- r_i, the radius of the i-th motor, z the distance from the center of mass of the quadrotor to the ground and h the vertical distance from the motor plane to the center of mass of the vehicle.
- ρ_g, a coefficient to adapt the ground effect from the quadrotor with respect to the helicopter literature.

Then, the thrust is modified according to (I.14):

$$T_i^g = \alpha_g(z) T_i, \tag{I.14}$$

with:

$$\alpha_g^{-1}(z) = 1 - \rho_g \left(\frac{r_i}{z+h} \right)^2. \tag{I.15}$$

It should be noted that when z is far away from zero, i.e. far from the floor, the coefficient (I.15) becomes closer to one: the ground effect has no more influence.

Blade flapping effect: The blade flapping effect, illustrated in Fig. I.7, is the result of the apparent wind effect during the motion of a motor-propeller assembly. The blade moving forward into the wind faces an apparent wind which increases its lift, the blade moving backward faces a tailwind which decreases its lift. Thus, the thrust, theoretically perpendicular to the plane of the propeller, is applied a certain rotation, in particular related to the flexibility of the propeller. On the quadrotor, this effect is accentuated because it is present on the four propellers.

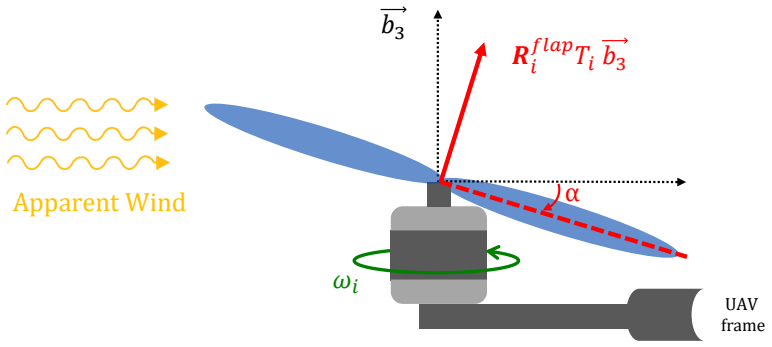


Figure I.7: Illustration of the blade flapping effect.

Several attempts to model the blade flapping effect have been proposed, in the following we will rely on the work of [Pro95] initially proposed for helicopters and the work of [Man15] adapted for quadrotors. It can be modeled by a rotation matrix (I.16) applied to each motor, given by:

$$\begin{aligned} \boldsymbol{R}^{flap} &= \boldsymbol{R}_x(b1s)\boldsymbol{R}_y(a1s), \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(b1s) & -s(b1s) \\ 0 & s(b1s) & c(b1s) \end{bmatrix} \begin{bmatrix} c(a1s) & 0 & s(a1s) \\ 0 & 1 & 0 \\ -s(a1s) & 0 & c(a1s) \end{bmatrix}, \end{aligned} \tag{I.16}$$

where $a1s$ and $b1s$ flapping angles [Pro95] are given by:

$$\begin{aligned} a_{1s} &= \frac{\left(\frac{8}{3}\theta_0 + 2\theta_1\right)\mu - C_T}{1 - \frac{\mu^2}{2}} + \frac{\frac{-16q}{\gamma\omega_r} + \frac{p}{\omega_r}}{1 - \frac{\mu^2}{2}}, \\ &= a_{1\mu} + a_{1\omega}, \\ b_{1s} &= \frac{\frac{4C_T}{3\sigma}}{1 + \frac{\mu^2}{2}}\left(\frac{2\mu\gamma}{3a} + \frac{\sigma}{2\mu}\right) + \frac{\frac{-16p}{\gamma\omega_r} - \frac{q}{\omega_r}}{1 + \frac{\mu^2}{2}}, \\ &= b_{1\mu} + b_{1\omega}. \end{aligned}$$

and with the following notations:

- $\theta_0[^\circ]$: pitch that the blade would have if extended to the center of rotation,
- $\theta_1[^\circ]$: is the twist angle,
- $a[-]$: slope of the lift curve per radian, assumed to be equal to 6,
- $\mu[-]$: is the tip speed ratio,

$$\mu = \frac{\|V_h\|}{\omega_r r} = \frac{\sqrt{V_{x,\mathcal{B}}^2 + V_{y,\mathcal{B}}^2}}{\omega_r r},$$

- $\sigma[-]$: represents the motor solidity ratio:

$$\sigma = \frac{n_{blade} c}{\pi r},$$

  with n_{blade} is the number of blades and c the width of the propeller (chord),

- $\gamma[-]$: is the Lock number,

$$\gamma = \frac{\rho a c r^4}{I_b},$$

  with I_b: the moment of inertia of the blade.

One can now express the thrust vector for each motor in the body frame. Individual thrust force vector is modified as follows:

$$T_i^{flap} = \boldsymbol{R}_i^{flap} T_i. \tag{I.17}$$

Using (I.17), it results in a new expression of the total force (I.9) given by (I.18):

$$F_T^{flap} = \sum_{i=1}^{4} T_i^{flap} \vec{b_3} = \sum_{i=1}^{4} \boldsymbol{R}_i^{flap} T_i \vec{b_3}. \tag{I.18}$$

To conclude with the balance of forces, we can write the sum of forces including weight, drag and thrust modified with the ground effect and the flapping effect, in the inertial frame:

$$\sum F = F_p + \boldsymbol{R} F_{Tg}^{flap} + F_v + F_{ext}. \tag{I.19}$$

Developing eq. (I.19), we obtain:

$$m\dot{v} = -mg\vec{i_3} + \alpha_g \boldsymbol{R} \sum_{i=1}^{4} \boldsymbol{R}_i^{flap} T_i \vec{b_3} - \boldsymbol{K_v}|v|v + F_{ext}. \tag{I.20}$$

I.4.2 Balance of torques

In this subsection, we introduce known torques that are applied to the rigid-body:

- Gyroscopic effect: Each motor may be considered as a rigid disc rotating around the vertical axis, the body frame. The motor's axis of rotation is itself moving with the angular velocity of the frame. This leads to the following torque, Γ_{gyro} given by (I.21):

$$\Gamma_{gyro} = \sum_{i=1}^{4} \boldsymbol{I_r}(\Omega \times \vec{b_3})\omega_i = \boldsymbol{I_r}(\Omega \times \vec{b_3}) \sum_{i=1}^{4} \omega_i, \tag{I.21}$$

  where $\boldsymbol{I_r}$ is the inertia matrix of a motor and $\times$ denotes for the cross product.

- Yaw torque: As the propeller is not perfect, it generates an additional drag (I.22). This drag causes a counter-torque, opposite to the direction of rotation of the propeller. It is not affected by the flapping effect.

$$\Gamma_D = \sum_{i=1}^{4} (-1)^{i+1} c_Q \omega_i^2 \vec{b_3}. \tag{I.22}$$

- Inertial counter-torque: It is generated by the residual acceleration of each propellers, due to individual rotation speed changes. It is given by (I.23):

$$\Gamma_{res} = \boldsymbol{I_r} \dot{\omega}_{res}^2 \vec{b_3}, \tag{I.23}$$

  where $\omega_{res} = -\omega_1 + \omega_2 - \omega_3 + \omega_4$.

- Torque induced by thrust:

$$\Gamma_T = \sum_{i=1}^{4} F_{T_i} \times P_i, \tag{I.24}$$

  with $\boldsymbol{P} = \begin{bmatrix} l' & l' & h \\ l' & l' & h \\ -l' & l' & h \\ -l' & -l' & h \end{bmatrix}$,

  where P_i refers to the i-th row of $\boldsymbol{P}$, $l' = \dfrac{\sqrt{2}}{2} l$, l is the arm length and h_{rotor} the height of the propeller in relation to the center of gravity. Now taking into account the flapping effect in (I.24), one gets the new expression (I.25):

$$\Gamma_T = \sum_{i=1}^{4} (\boldsymbol{R_i^{flap}} F_{T_i} \times P_i) = \sum_{i=1}^{4} (c_T \omega_i^2 \boldsymbol{R_i^{flap}} \vec{b_3} \times P_i). \tag{I.25}$$

- Disturbance torque Γ_{ext} that includes other external unknown disturbances. It can for example include wind induce torque.

Considering first that there is no flapping effect, the torque vector can be calculated as:

$$\Gamma = \Gamma_T + \Gamma_D = \begin{bmatrix}\Gamma_p \\ \Gamma_q \\ \Gamma_r\end{bmatrix} = \begin{bmatrix}-c_T l' & c_T l' & c_T l' & -c_T l' \\ -c_T l' & -c_T l' & c_T l' & c_T l' \\ c_Q & -c_Q & c_Q & -c_Q\end{bmatrix} \begin{bmatrix}\omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2\end{bmatrix}.$$

One recalls Euler's rotation equation, in the inertial frame, using L the angular momentum and the total applied torque $\sum \Gamma$:

$$\left.\frac{\mathrm{d}L}{\mathrm{d}t}\right|_{\mathcal{I}} = \sum \Gamma \tag{I.26}$$

As $L = \boldsymbol{J}\Omega$ is expressed in the body-fixed frame, and it is rotating at Ω, we use the transport theorem to express (I.26) in the body frame:

$$\left.\frac{\mathrm{d}L}{\mathrm{d}t}\right|_{\mathcal{I}} = \left.\frac{\mathrm{d}L}{\mathrm{d}t}\right|_{\mathcal{B}} + \Omega \times \boldsymbol{L} = \boldsymbol{J}\dot{\Omega} + \Omega \times \boldsymbol{L} = \sum \Gamma \tag{I.27}$$

To conclude with the balance of torques, we can sum up all contributions listed above and using (I.27):

$$\sum \Gamma = \boldsymbol{J}\dot{\Omega} + \Omega \times \boldsymbol{J}\Omega = \Gamma_{gyro} + \Gamma_{res} + \Gamma_T + \Gamma_D + \Gamma_{ext}. \tag{I.28}$$

Let's define $\Omega_\times$ the skew-symmetric matrix associated to Ω. It that satisfies, for all matrix $\boldsymbol{M} \in M_3(\mathbf{R})$, $\boldsymbol{M} \times \Omega = \boldsymbol{M}\boldsymbol{\Omega}_\times$:

$$\boldsymbol{\Omega}_\times = \begin{bmatrix}0 & p & -q \\ -p & 0 & r \\ q & -r & 0\end{bmatrix}. \tag{I.29}$$

Developing all terms of eq. (I.28) and using notations (I.29), one gets:

$$\begin{aligned}\boldsymbol{J}\dot{\Omega} \;=\; & -\boldsymbol{\Omega}_\times \boldsymbol{J}\Omega + \boldsymbol{I_r}(\Omega \times \vec{b_3}) \textstyle\sum_{i=1}^4 \omega_i + \sum_{i=1}^4 (c_t \omega_i^2 \boldsymbol{R_i^{flap}} \vec{b_3} \times P_i) \\ & + \textstyle\sum_{i=1}^4 (-1)^{i+1} c_Q \omega_i^2 \vec{b_3} + \boldsymbol{I_r} \dot{\omega}_{res}^2 \vec{b_3} + \Gamma_{ext}.\end{aligned} \tag{I.30}$$

Now that we have established the balance of forces and torques, we can derive the complete model of the quadrotor.

I.4.3 Complete quadrotor modeling

According to balances of forces and torques, respectively given in equations (I.20) and (I.30), one can establish the complete model of the quadrotor dynamics.

Quadrotor complet model, using Newton-Euler formalism:

$$\begin{cases} \dot{\xi} &= v, \\ m\dot{v} &= -mg\vec{i_3} + \alpha_g \boldsymbol{R} \sum_{i=1}^{4} \boldsymbol{R}_i^{flap} T_i \vec{b_3} - \boldsymbol{K_v}|v|v + F_{ext}, \\ \dot{\boldsymbol{R}} &= \boldsymbol{R}\boldsymbol{\Omega}_{\times}, \\ \boldsymbol{J}\dot{\Omega} &= -\boldsymbol{\Omega}_{\times}\boldsymbol{J}\Omega + \boldsymbol{I_r}(\Omega \times \vec{b_3}) \sum_{i=1}^{4} \omega_i + \sum_{i=1}^{4} (c_t \omega_i^2 \boldsymbol{R_i^{flap}} \vec{b_3} \times P_i) \\ & \quad + \sum_{i=1}^{4} (-1)^{i+1} c_Q \omega_i^2 \vec{b_3} + \boldsymbol{I_r} \dot{\omega}_{res}^2 \vec{b_3} + \Gamma_{ext}. \end{cases} \tag{I.31}$$

This model includes as much as possible of known effect applied on the quadrotor. It is highly nonlinear. It can be complicated to derive control algorithms from it. The quadrotor complete model will be mainly used in simulation to firstly model the quadrotor and to test control algorithms before experimenting real flights.

I.5 Quadrotor simplified model

In the previous section, we have tried to model the quadrotor as precisely as possible by giving its complete model. This model is complex and requires the knowledge of many parameters to completely describe the quadrotor's motions. These parameters are often difficult to obtain or estimate. It becomes difficult to derive control algorithm from (I.31). This model is suitable for simulation purposes and testing control algorithm.

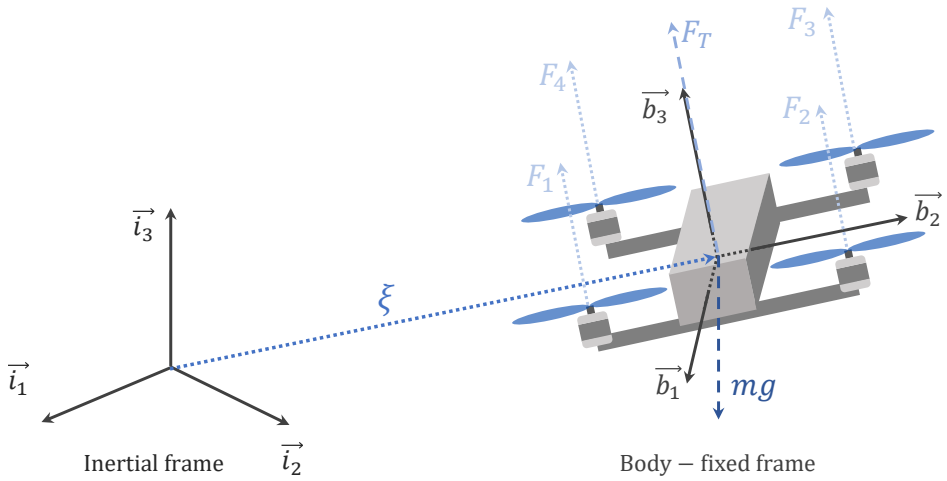


Figure I.8: Quadrotor mains forces after making assumptions.

To design a controller, a simpler, nonlinear, model is very often used. In order to obtain the commonly used model in the literature, we must first make several simplifications and assumptions. Basic assumptions and hypothesis done are as follows:

- Flying away from the ground: the ground effect no longer operates $\alpha_g = 1$,
- The quadrotor is no flying at high velocity: the drag force is lessen $F_v = \mathbb{0}_3^T$, where $\mathbb{0}_3 := [0,\ 0,\ 0]$,
- Motors are not flexible: the blade flapping is weak $\boldsymbol{R_i^{Flap}} = \mathbf{0_3}$,
- There is no residual acceleration: $\Gamma_{res} = \mathbb{0}_3^T$,
- The gyroscopic effect is negligible: $\Gamma_{gyro} = \mathbb{0}_3^T$.

Before deriving the simplified model one can give the following relation:

$$\begin{bmatrix} T \\ \Gamma_p \\ \Gamma_q \\ \Gamma_r \end{bmatrix} = \begin{bmatrix} c_T & c_T & c_T & c_T \\ -c_T l' & c_T l' & c_T l' & -c_T l' \\ -c_T l' & -c_T l' & c_T l' & c_T l' \\ c_Q & -c_Q & c_Q & -c_Q \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix}. \tag{I.32}$$

In equation (I.32), we notice the mixing matrix between the squared motor speeds and forces & moments. This matrix depends on the drone configuration, here it is expressed for a $X4$-configuration (I.1a).

The equations of quadrotor dynamics given in (I.31) are simplified using previously established hypothesis and eq. (I.32). The standard model can be derived, as follows.

Quadrotor simplified model, using Newton-Euler formalism:

$$\begin{cases} \dot{\xi} &= v, \\ m\dot{v} &= -mg\vec{i_3} + \boldsymbol{R}T\vec{b_3}, \\ \dot{\boldsymbol{R}} &= \boldsymbol{R}\boldsymbol{\Omega}_\times, \\ \boldsymbol{J}\dot{\Omega} &= -\boldsymbol{\Omega}_\times \boldsymbol{J}\Omega + \Gamma. \end{cases} \tag{I.33}$$

This dynamical model is used by several authors such as Mahony *et al.*[MKC12], Bouabdallah *et al.*[BMS04] or Manecy [Man15]. It is very often used to model and design linear and nonlinear controllers for quadrotors. As the mixing matrix is non-singular, we can chose the thrust and torques as command inputs for the quadrotor system. Then, using computed commands one can retrieve angular speed of each motor to achieve desired flights.

Now that the nonlinear models of the quadrotor are derived, the next chapter will give more details about the experimental platform and the control architecture used to pilot it.

Chapter II

Experimental setup

This chapter presents the experimental setup used to assess flight performance of the proposed algorithms. It introduces the quadrotor experimentally used and the motion capture room. It then describes the structure commonly used for quadrotor piloting: the cascaded architecture. It also gives more technical insights about the software used and how data are collected and processed, using the communication protocols established with ROS.

Contents

II.1 Introduction

This chapter is a development of the experimental aspect of this thesis. It first introduces the quadrotor used by describing its components and their uses in the section II.2. The experimental motion capture room and the setup used for the experiments is then established in section II.3.

After a brief explanation of the Robot Operating System (ROS) functioning, an explanation of the communication system using topics and messages is given in section II.4.

Section II.5 is dedicated to the quadrotor cascaded control architecture. The control architecture used by PX4 is first introduced. It will allow us to recall the architecture scheme commonly used for the control of quadrotor. Once done, one introduces the general architecture used in the remainder of this manuscript and used for both simulation and experiment. This chosen architecture is based on the previous one but presents some differences which will be detailed. The selected architecture will rely on several PX4 elements. The latter performs the low-level angular control. Thus, explanations of the essential components of PX4 are given:

- Proposed solution will depend on the angular velocity loop. We will explain how we have tuned the PID parameters for the angular velocities.
- We describe how our control command architecture communicates with PX4 low-level control loop. A brief description of how computed commands are scaled and relayed to the inner loop of PX4 is also given.
- PX4 uses a particular mixer, different from the mixer given in eq. (I.32). In order to better understand and establish a simulator of the quadrotor used, we have identified this block. Explanation of the identification is given here.
- Time pure delay can be observed in the loops. A solution to better handle it is the use of smith predictor. Short explanations of it are provided.

Finally, some insights about data pre-processing are given in section II.6. Indeed, to perform control or learning, data coming from different sensors are used and can be very noisy. Explanations on data filtering using Savitzky-Golay are given.

II.2 The Holybro Kopis 2 quadrotor

In order to experimentally test and validate the proposed flight controllers developed in the following chapters, we have used a *Holybro Kopis 2* quadrotor. A picture of the device is given in Fig. II.1. It is a high-performance first-person view racing drone. Main specifications of this drone are given in Table II.1. The quadrotor is powered by 3S LiPo (Lithium Polymer) batteries.

Figure II.1: The Holybro Kopis 2 quadrotor.

The hardware of the device consists of:

- A **flight controller**. It includes an internal Inertial Measurement Unit (IMU) and a barometer. Here, it is a *Kakute F7* controller running along PX4 firmware [MHP15]; [Noa]. It is used as low-level controller, it performs angular rates control using PID for each component. The angular rate PID tuning is explained in section II.5.3. It is also used for state estimation using an Extended Kalman Filter (EKF), the ECL EKF2. Because of memory limitations, an additional on-board micro-controller has been implemented on the quadrotor. It is a *Orange-pi* controller. One uses it to perform proposed positions & velocity control and attitude control. This controller publishes computed commands either in {rates+thrust} or in {angles+thrust} to the Kakute F7. In the latter case, the attitude control is performed by PX4.
  As we are using deep neural networks, real-time predictions are performed on this controller. To do so, one uses TensorFlow Lite [Aba+15] which is known for fast on-board neural predictions.

- An **Electronic Speed Controller (ESC)**. It drives the four brushless motors of the quadrotor. It receives the signals from the Kakute F7 controller and uses them to adapt the power delivered to each motor. The protocol used is the Pulse Width Modulation (PWM). For the Kopis quadrotor, these are open loop ESC: there is no feedback to control rotational speed of the motors.

- Additional **sensors**. One can list: flow sensor, range sensor and a barometer.

- A **WiFi module**. Here, one uses the ESP32 WiFi module. It establishes a WiFi bridge

between the ground station computer and the flight controller. It communicates with the Kakute F7 using UART protocol.

- A **Remote Control (RC)** receiver. It is linked to a remote controller for manual control. The remote controller allows to arm or disarm the device and to switch between flight modes.

A schematic view of the hardware configuration is given in Fig. II.2.

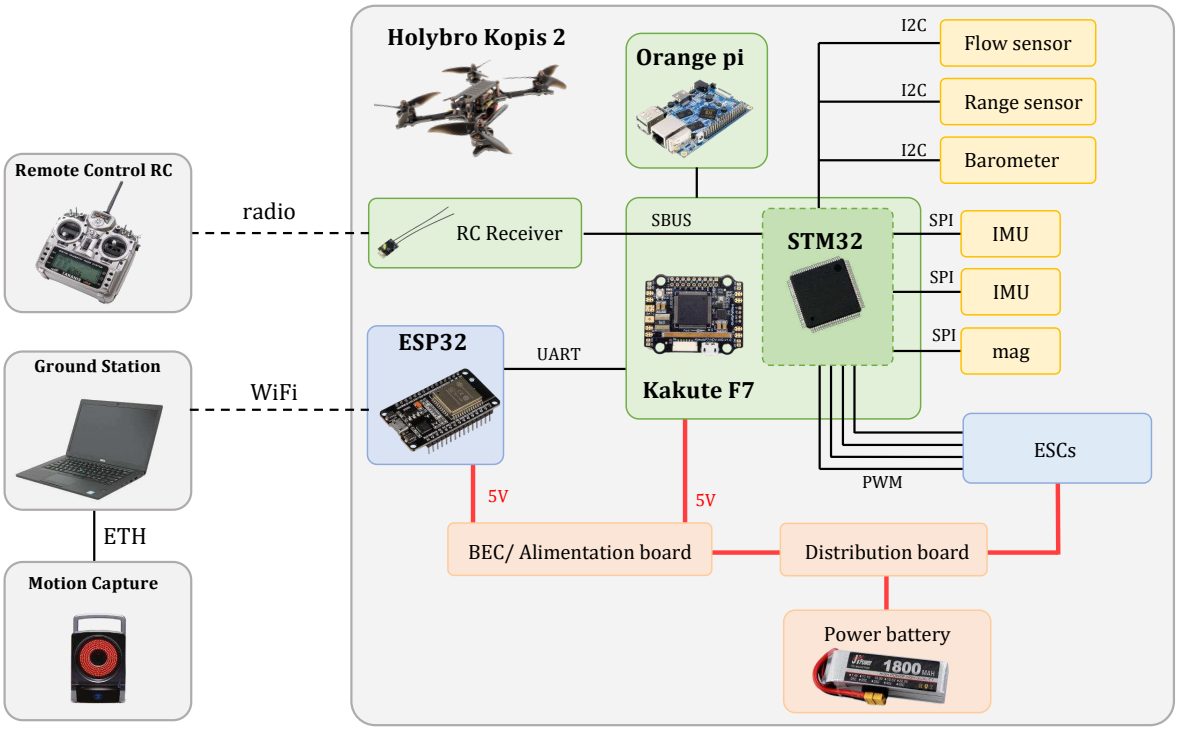


Figure II.2: The Holybro Kopis 2 hardware configuration.

For mission situations, the ground station sends the reference trajectories to the quadrotor. The station uses the WiFi bridge established by the ESP32 to communicate. The communication exchanges are done at a frequency of 100Hz. The station also receives spatial and angular positions information from the motion capture system installed in the experimental room. The room is presented in the next section II.3. A schematic representation of the entire experimental setup is shown in Fig. II.3. The trajectory generation is done before the mission. The user chooses the type of trajectory that the quadrotor will perform: position steps, circles, spirals, etc. Then, these instructions are generated and sent to the quadrotor. The trajectory generation is performed at 50Hz.

Attribute	Value of the attribute
MCU	STM32F745VGT6 32-bit processor
IMU	MPU6000 (SPI)
Barometer	BMP280
Weight (no battery)	325 grams
Battery	3S LiPo battery

Table II.1: The Holybro Kopis 2 specifications.

II.3 Motion capture room

All presented experimental tests are performed indoor using a Motion Capture (MOCA) system. The room is equipped of twelve *Vicon T40s* cameras. They track markers attached to the quadrotor. Spacial position and orientation of the device are then obtained. A dedicated computer uses *Vicon Tracker* software to track the quadrotor position using all camera signals. It communicates that computed position to the ground station computer through the Virtual Reality Peripheral Network (VRPN) protocol. An illustration of the room is given in Fig. II.3 and some pictures are provided in Fig. II.4.

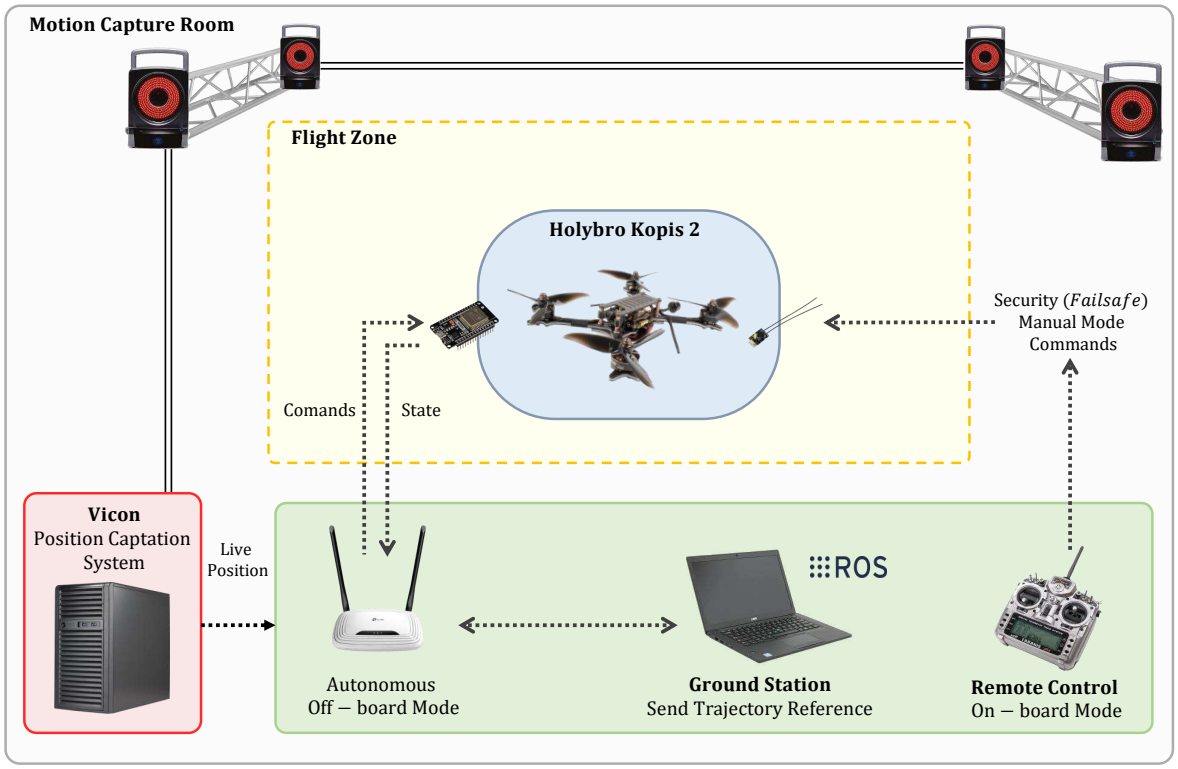


Figure II.3: Schematic representation of the motion capture room.

For safety and security, a remote controller is used to switch between different flight modes: arm/disarm, onboard/off-board mode. It is basically a remote control unit that sends commands, thanks to a transmitter, to the quadrotor that receive them via a receiver. The RC allows a manual retrieval of commands if there is a communication issue between the station computer and the device.

(a) MOCA flight zone (b) MOCA ground station

Figure II.4: Pictures of the motion capture room.

II.4 Robot Operating System

II.4.1 General overview

Robot Operating System (ROS) [Qui+09] is a robotic middleware framework. It is a flexible environment, open-source and free. It is composed of very useful libraries and tools for robot applications. ROS provides several functionalities of which the main one is to allow communication between different processes and machines. For instance, in our case, ROS enables the communication between the quadrotor, the ground station and the motion capture system. It has a record structure, named *rosbag*, which contains exchanged data. Using rosbag, recorded data can be analyzed or played back. ROS also integrates *Gazebo*, a simulation environment [KH04].

ROS allows communication between different executable code distributed between computers and robots. Main communication components used are:

- ROS *nodes* are executables that perform computations and tasks. They communicate between them using ROS *messages* through topics. They can either publish or subscribe to topics to send or receive information. A specific node type, called ROS Master, is used to establish communication between node. It provides naming and registration services to all nodes. The ROS Master helps nodes to find each other by tracking publishers and subscribers to topics.

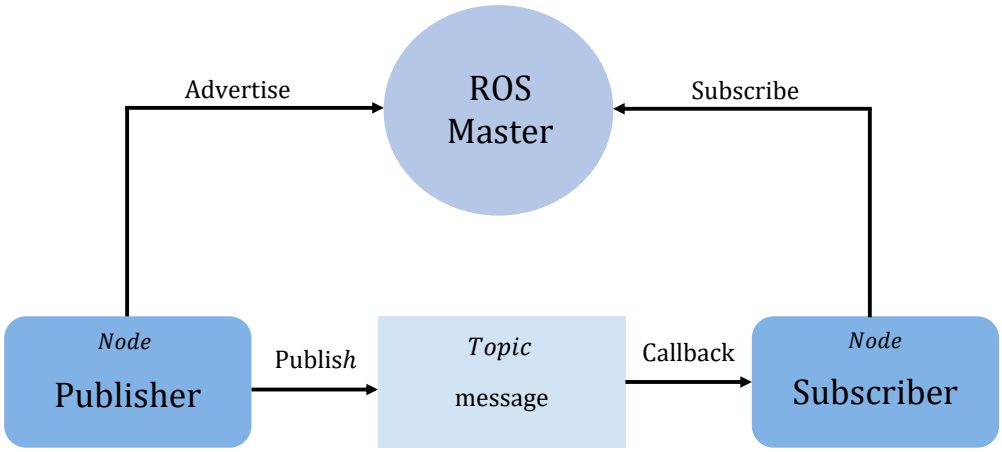


Figure II.5: ROS publisher/subscriber communication mechanism. Nodes communicate a message through a given topic. They can either subscribe or publish to topics. Nodes can do both for multiple topics.

- ROS *topics*: are information buses that allow nodes communication. Data communicated through topics are defined by ROS messages. A node can subscribe to a topic in order to receive messages via subscribers. It can publish to a topic to send messages via publishers.
- ROS *messages*: are data type. It exists a large variety of data including basic one as float, string, boolean. Messages can be even more complex including headers and other message types.
- ROS *services*: are an alternative way to communicate data between nodes. It allows request / reply interactions for distributed systems. A ROS service is a synchronous system and it is defined by a name and a pair of messages.

The publisher/subscriber communication mechanism of ROS is illustrated in Fig. II.5. It represents a very simple communication process between two nodes: one publisher and one subscriber. This is at the basis of ROS functioning.

ROS is developed in two languages: C++ and Python. As one uses deep neural networks and because Python has a very well-developed, open-source, documented packages for learning, we decided to develop our algorithms in Python.

A very important ROS package used for communication is *mavros*. It enables *MAVLink* communication between the ground station computer and the quadrotor. *MAVLink* is a lightweight communication protocol to communicate with drones or for internal drone communication.

II.4.2 ROS topics for the controller node

As previously stated, data messages are communicated through different topics. Our flight controller will need information from different sensors, for instance, the state composed of the quadrotor positions and velocities. It therefore needs information coming from different topics in order to properly work. Thus, the control node will need to subscribe to different topics, whose main ones are listed below:

- */mavros/local_position/odom*. This topic contains the spatial pose (position and orientation) and velocities of the quadrotor computed by the EFK2. Provided velocities are given in the body frame.
- */mavros/vision_pose/pose*. It allows to get spatial position of the quadrotor from the Vicon cameras. Actually, it is a remap of the */vrpn_client_node/Kopis_pose/pose*. The VRPN client reads motion capture data and communicates it via this topic.
- */mavros/states_references*. It contains the references trajectories computed on the ground station and sent to the drone.
- */mavros/imu/data*. It allows to communicate data collected by the IMU: angular velocities, linear accelerations and orientation. This topic contains filtered data. A raw version exists, providing original signals but consequently they are more noisy.
- */mavros/battery*. It provides information on the battery status. A particularly interesting information, that we will use later, is the battery voltage which is an image of the state of charge of the battery.

Concerning the controller node, once the commands are computed, it will need to send them to the flight controller for low-level control. It is thus needed to publish commands in a dedicated topic. Here, *mavros/setpoint_raw/attitude* is the topic that makes it possible. More details of how this topic is used are given in section II.5.4.

The data exchanged via rostopics can be saved and stored using *rosbags*. As ROS links several nodes from different sources, the recorded data is saved at its own frequency. For example, the position coming from the Vicons (*/mavros/vision_pose/pose*) is published at 100Hz whereas the battery status (*/mavros/battery*) is given at 5Hz. As far as learning is concerned, this is an issue. Indeed, the database must contain a coherent data matrix regarding temporal samples. To solve the problem, we propose to use data at the same frequency using a personalized ROS message, called *statecommand.msg* and communicated through a dedicated topic. An illustration of this custom message is given in Fig. II.6. It contains all the measured and estimated variables necessary to the controller design of the drone and to our learning approaches presented in this manuscript. This message is built using information from previously mentioned topics:

- linear positions, from the Vicons and from the EKF,
- angular positions (quaternion) and Euler angles, from the EKF and the Vicon,

- expected linear positions[1],
- references in positions and yaw,
- linear velocities from the EKF, in inertial and body frame,
- linear accelerations from the IMU,
- angular velocities,
- battery voltage,
- the commands send (thrust, angles or angular rates),

This message is filled within the controller node. For convenience, it is published at the same frequency as the position and attitude control loop, at 100Hz. As everything is collected at 100Hz, some data such as the state references or the battery voltage will be over-sampled. Thus, each data coming from different topics will have the same number of elements. It will be convenient for creating the training and validation database for the learning procedure.

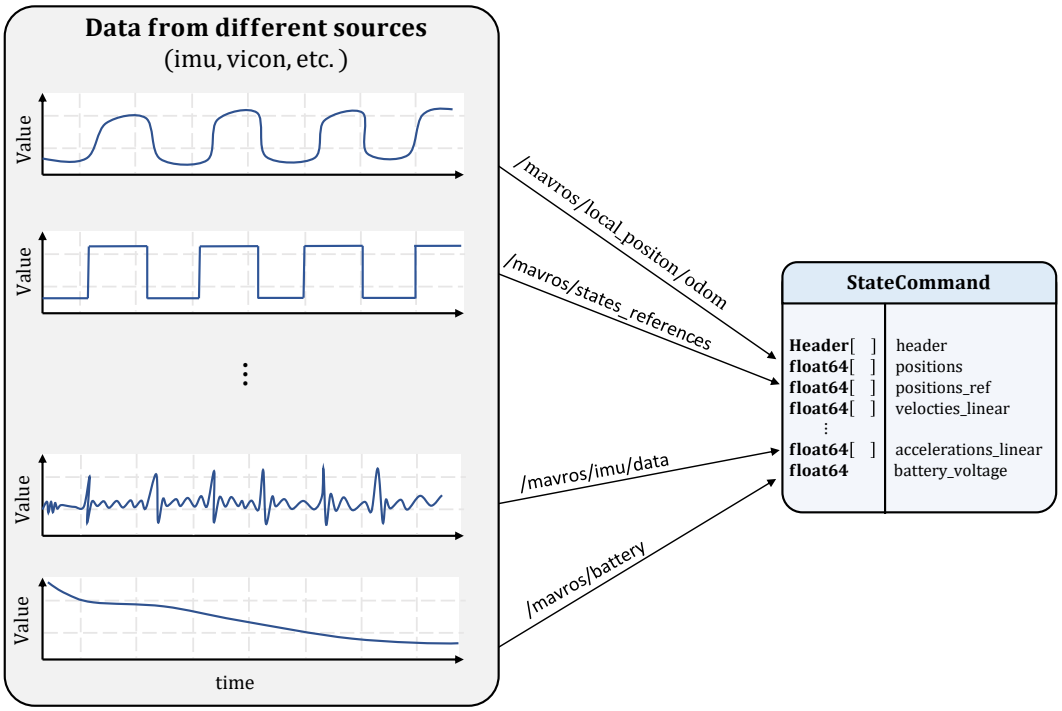


Figure II.6: The *statecommand.msg*: a custom message to collect all the relevant data at the same time stamps. It is built using message from all listed topics at each control step.

Now that overall ROS structure and data communication is explained, we can go deeper in understanding the controller architecture that composes the quadrotor autopilot.

[1]The expected linear positions are computed according to a given linear model on which is based the linear control. More details of this information will be given on next chapters.

II.5 Quadrotor cascaded control architecture

This section gives the basis of the cascaded control architecture used for quadrotor control applications. We will rely on PX4 autopilot which gives a first picture of such control scheme for industrial multicopters. The proposed architecture in this manuscript will be greatly inspired of it. A general overview of the control scheme adopted in the following of the present work will be then given. Mathematical and theoretical details about proposed controllers will be explained in each corresponding chapters (III, IV and V). Next, as we will rely on PX4 for several features we will explain them. It deals with tuning angular loop, understanding used mixers and how commands computed are provided to the low-level controller.

II.5.1 PX4 controller architecture

II.5.1.1 PX4 autopilot overview

PX4 autopilot is an open-source autopilot software for autonomous robots mainly focused on aircraft systems. Quadrotor developed at GIPSA-lab are using it since it provides different level of drone control and safety features. If something fails, it triggers a failsafe mode. For example, a low battery level, an off-board loss, a position loss, etc. can trigger this mode. The failsafe action can then be chosen such as a landing procedure, holding the current position or return to home, the RTH mode.

II.5.1.2 PX4 cascaded controller

PX4 uses a very common control architecture: a cascaded P(ID) scheme to control successive loops. It is composed of four cascaded controllers: position (P), velocity (PID), attitude (P) and angular rate (PID) control loops, as represented in Fig. II.7. The gains of each loop must be correctly tune to achieved desired performance for each component. It is also important to notice that each inner loop must be fast enough with respects to its respective outer-loop to work properly.

It should be noted that between velocity control and angle control a conversion block is used in that scheme. This block links desired accelerations from the velocity controller to a desired attitude for the angular control. A manner to design this conversion block is to use linear accelerations equations from (I.33), to get a relation between variables of interest.

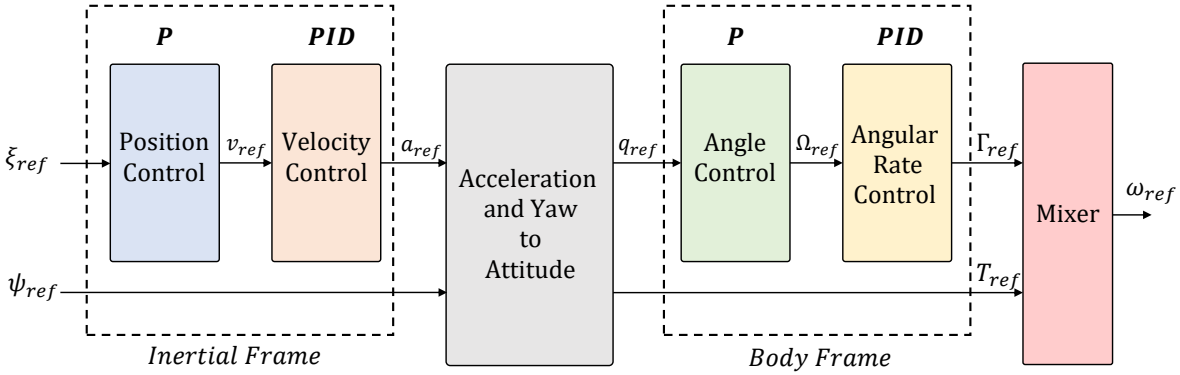


Figure II.7: Common cascaded control architecture for quadrotor.

Developing it on each axis, we get:

$$\begin{cases} \ddot{x} &= \dfrac{T}{m}(c_\phi s_\theta c_\psi + s_\phi s_\psi), \\ \ddot{y} &= \dfrac{T}{m}(c_\phi s_\theta s_\psi - s_\phi c_\psi), \\ \ddot{z} &= -g + \dfrac{T}{m} c_\phi c_\theta. \end{cases} \tag{II.1}$$

Assuming that the drone is moving with small angles, equations (II.1) can be expressed as:

$$\begin{cases} \ddot{x} &= \dfrac{T}{m}(\theta c_\psi + \phi s_\psi), \\ \ddot{y} &= \dfrac{T}{m}(\theta s_\psi - \phi c_\psi), \\ \ddot{z} &= -g + \dfrac{T}{m} c_\phi c_\theta. \end{cases} \tag{II.2}$$

From (II.2), a relation can be deduced between desired thrust, roll and pitch and the linear accelerations:

$$\begin{cases} \theta &= \arctan\left(\dfrac{\ddot{x}c_\psi + \ddot{y}s_\psi}{\ddot{z}+g}\right), \\ \phi &= \arctan\left(c_\theta \dfrac{\ddot{x}s_\psi - \ddot{y}c_\psi}{\ddot{z}+g}\right), \\ T &= m\dfrac{\ddot{z}+g}{c_\phi c_\theta}. \end{cases} \tag{II.3}$$

Using expressions (II.3) is a way to implement that *acceleration to attitude* block. It is not an exact relation but it works correctly for very common flights. Other solutions exist to implement that conversion.

For more acrobatic flights or other applications, there exists many ways to perform attitude and position control. Regarding our present study, we focus on the cascaded architecture, shown above. We based our work on that architecture. However, we adapted it in order to remove the conversion block and provide directly reference angles to the attitude loop.

II.5.2 Implemented controllers architecture

The architecture chosen for our applications is directly inspired by the standard cascaded control architecture presented as before. This choice comes from several reasons. First, to keep the simplicity of tuning of such architecture. Second, to ease the adaptation of developed controllers and learning solutions on any quadrotor. In that manner, it will be easier to implement such algorithms in several standard quadrotors without the need of excessively technical changes. Indeed, it is more straightforward to either change position/velocity or attitude control blocks, without replacing everything.

For experimentation and implementation reasons, we developed our own position, velocity and attitude controllers. We do not focus on angular rate algorithm design. However, to correctly setup the quadrotor, it is required to obtain adequate gains of the PID piloting the angular rates. The tuning process is explained in the next sub-section.

In the proposed scheme, we combined position and velocity control in a same control loop. It provides references in thrust and angles to the attitude loop. The low-level control is performed by PX4. The general architecture we implemented is summarized in Fig. II.8.

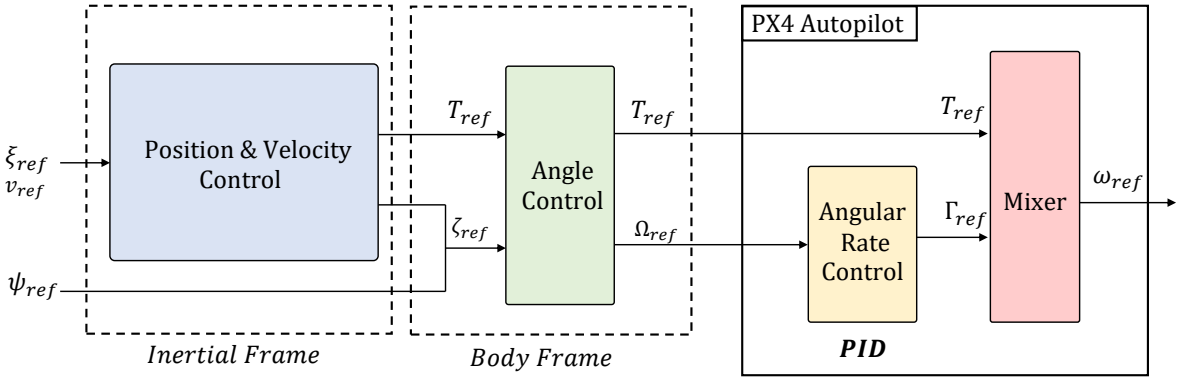


Figure II.8: Proposed general control architecture.

A notable difference, compared to the previously presented architecture II.5.1.2, is that we do not rely on a conversion block to convert accelerations to attitude. Indeed, this is an inexact block, made with simplifying assumptions therefore adding some errors in the whole scheme. In this manner, our position and velocity control directly provides desired angles and thrust for the inner loop. More details about control laws, learning-based control and observers will be given in corresponding chapters.

II.5.3 Angular rate PID tuning

To pilot the pitch, roll and yaw rates to the desired values, PX4, like many industrial and open-source solutions, uses PID controllers for each component. Before being able to set up the outer-loops, it is necessary to have the inner-loop running faster and correctly. As we will rely on that control architecture, a good tuning is crucial. Thus, the first step is to setup each PID gains for fast and stable rate converge.

Let's first introduce the PID controller in its standard form (II.4):

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau)\mathrm{d}\tau + k_d \frac{\mathrm{d}e(t)}{\mathrm{d}t}, \tag{II.4}$$

where x is the measured value of the regulated variable and x_{ref} is the desired value for x and where:

- k_p: is the proportional gain,
- k_i: is the integral gain,
- k_d: is the derivative gain,
- $e(t) = x(t) - x_{ref}(t)$: is the error term.

However, PX4 uses a slightly different PID form (II.5). Because of the derivative of the reference, a large initial command peak can occur. That peak is also referred as *derivative kick* or *setpoint kick*. A possible solution is to take the derivative term on the feedback value instead of the error. It is a particular configuration of PID with *setpoint weighting* [ÅM21], where the weighting factor is set to zero, see Fig. II.9. The expression of the PID used is given by:

$$u(t) = k(k_p e(t) + k_i \int_0^t e(\tau)\mathrm{d}\tau) + k k_d \frac{\mathrm{d}x(t)}{\mathrm{d}t}. \tag{II.5}$$

This mathematical expression allows a mixed implementation of parallel and standard form for that PID. The use of $k_p = 1$ allows to get the standard form while $k = 1$ provides the parallel form.

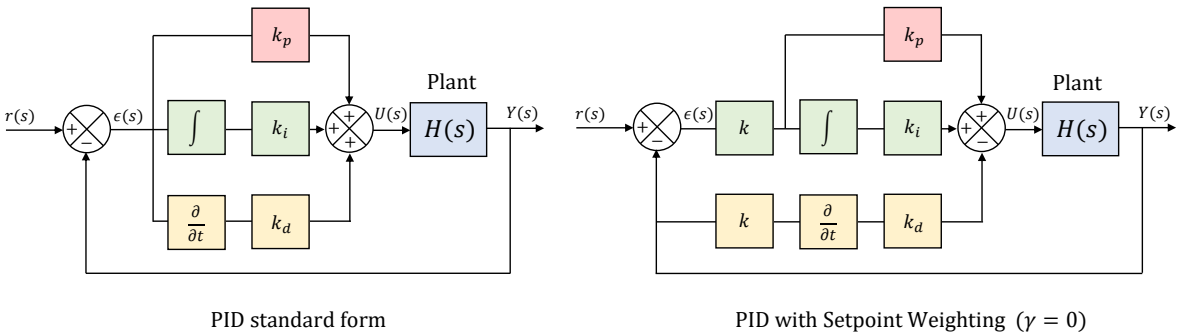


Figure II.9: Different PID architectures. (Left) Standard PID form. (Right) PID with weighting setpoint, the weighting is set to $\gamma = 0$, meaning the derivative operate only on the feedback.

Here, one considers angular velocities given by $\Omega = \begin{bmatrix} p, & q, & r \end{bmatrix}^T$. For each component is associated a PID controller. Taking the standard form, setting $k = 1$, it results in the tuning of nine gains. As angular rate loop tuning is not easy to perform in practice, we performed it using Gazebo and Matlab simulation environments by doing step over each individual angular rate. Indeed, performing true steps in angular velocity is a very risky and challenging task. To perform it in an experimental manner, a test bench on which the quadrotor is attached with non-slip bearings is needed.

There are several methods to adjust PID gains, we can cite the well known *Ziegler-Nichols* method [ZN93], that can be used both for open-loop and closed-loop systems. This method results in overshoots, that for our application we tend to cancel. Consequently, we have chosen a manual adjustment of the PID gains. First k_p is increased setting k_i and k_d to zero or very small. It will speed up the system response but can lead to oscillations. Then tune k_i by increasing it to remove static error. It must not be too high because it causes oscillations and can lead to instability. Finally, k_d is used to attenuate oscillations but it will slower the response. After several simulation and experimental adjustments, it results in the gains provided in Table II.2.

	Roll rate	Pitch rate	Yaw rate
k	1.0	1.0	1.0
k_p	0.085	0.085	0.15
k_i	0.32	0.32	0.1
k_d	0.0008	0.0008	0.0

Table II.2: PID gains for PX4 angular rate loop.

II.5.4 PX4 command

The topic used to send command to the quadrotor is */mavros/setpoint_raw/attitude*. The message communicated through this topic is an *AttitudeTarget* message from Mavros. It allows to either pilot desired angles or desired angular rates. If piloting in attitude, the message requires desired quaternion, we recall the conversion to perform to get quaternions from Euler angles, with (II.6):

$$\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}_{des} = \begin{bmatrix} c_{\phi/2}c_{\theta/2}c_{\psi/2} + s_{\phi/2}s_{\theta/2}s_{\psi/2} \\ s_{\phi/2}c_{\theta/2}c_{\psi/2} - c_{\phi/2}s_{\theta/2}s_{\psi/2} \\ c_{\phi/2}s_{\theta/2}c_{\psi/2} + s_{\phi/2}c_{\theta/2}s_{\psi/2} \\ c_{\phi/2}c_{\theta/2}s_{\psi/2} - s_{\phi/2}s_{\theta/2}c_{\psi/2} \end{bmatrix}_{des} \tag{II.6}$$

The control mode selection, either angular or attitude control, is made via the use of a mask included in the message. By selecting *type_mask = 0b10000000* (128), PX4 controller performs the angular control. The default mask, is automatically selecting the attitude control mode.

The thrust to be provided to PX4 must be scaled between 0 and 1, where 1 corresponds to the maximum throttle that the quadrotor can deliver. An important parameter must be determined before flying, here called $hover_{comp}$. It corresponds to the throttle value to get the quadrotor hovering.

The normalization performed (II.7) is given by the following relation:

$$thrust = thrust_{ref} \cdot \frac{hover_{comp}}{mg} + hover_{comp} \cdot offset_{lin}, \tag{II.7}$$

where $offset_{lin}$ is equal to one if $thrust_{ref}$ is computed around the equilibrium given by mg or 0 otherwise.

II.5.5 PX4 mixer identification

The autopilot proposed by PX4 directly maps motor speeds w_i to PWM values. As it is difficult to explicitly control what is performed by the algorithm, we decided to carry out an identification experiment to find the linear mapping used by PX4 between commands and PWM values, of the form (II.8):

$$PWM = mixer_{PX4} \cdot U + offset_{PX4}, \tag{II.8}$$

where U is the command such that $U = \left[\Gamma_p,\ \Gamma_q,\ \Gamma_r,\ T\right]^T$.
Let $Y^i = \left[PWM_1^i,\ PWM_2^i,\ \ldots,\ PWM_N^i\right]$ be N observations of the i-th PWM and X be the *regressor matrix*:

$$X = \begin{bmatrix} 1 & U_{1,1} & U_{1,2} & U_{1,3} & U_{1,4} \\ 1 & U_{2,1} & U_{2,2} & U_{2,3} & U_{2,4} \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & U_{N,1} & U_{N,2} & U_{N,3} & U_{N,4} \end{bmatrix}.$$

Next, introducing ϵ^i as the error term vector for the i-th PWM, we need to solve for each i:

$$Y^i = X \begin{bmatrix} a_1^i \\ a_2^i \\ a_3^i \\ a_4^i, \end{bmatrix} + \epsilon^i,$$

where the error term is described as $\epsilon^i = \begin{bmatrix} \epsilon_1^i \\ \epsilon_2^i \\ \vdots \\ \epsilon_N^i. \end{bmatrix}$.

To perform the identification, we first built the regressor matrix using data. One consequently produces flight data and recorded them using rosbags. The data of interest is

the command output from PX4 provided via the topic */mavros/target_actuator_control* and PWM values stored in */mavros/rc/out*.

The least squares method can be used to find a solution. It is given by (II.9):

$$\hat{\beta} = (X^T X)^{-1} X^T Y^i. \tag{II.9}$$

One obtains the following mixer:

$$mixer_{PX4} = \begin{bmatrix} -493.6 & 706.4 & 769.8 & 1000.5 \\ 494.0 & -705.2 & 997.6 & 1001.3 \\ 494.8 & 705.2 & -762.2 & 1001.9 \\ -495.1 & -706.4 & -1003.1 & 997.0 \end{bmatrix},$$

and the following offset:

$$offset_{PX4} = \begin{bmatrix} 999.3 \\ 999.1 \\ 998.9 \\ 1000.4 \end{bmatrix},$$

with the an R^2 coefficient of 0.997.

Using this information, we were able to refine our Matlab/Simulink quadrotor simulator. The idea behind this simulator is to reproduce PX4 and Gazebo within a unique simulation structure that we totally master. By that, we are able to add effects modeled in chapter I. It is also faster and easier to perform simulations using Matlab. In addition, the tuning gain process is made faster.

II.5.6 Smith predictor

The experimental implementation of the algorithms has highlighted the presence of a pure delay in the WiFi communication. A manner to manage this delay is the use of a *Smith predictor*. The idea behind this controller is to synthesize an initial controller for the process without delay and then propose a controller adapted to the delayed system based on the first one.

Let $H(z)$ be the system model and z^{-k} be the pure time delay. First considering only H, one designs a controller $C(z)$ giving expected closed-loop behavior:

$$H_{CL} = \frac{C(z)H(z)}{1 + C(z)H(z)}.$$

The corrected controller that includes the pure time delay is given by:

$$C_k = \frac{C(z)}{1 + C(z)H(z)(1 - z^{-k})}.$$

The structure of the Smith predictor filter is illustrated in Fig. II.10. To illustrate, the effect

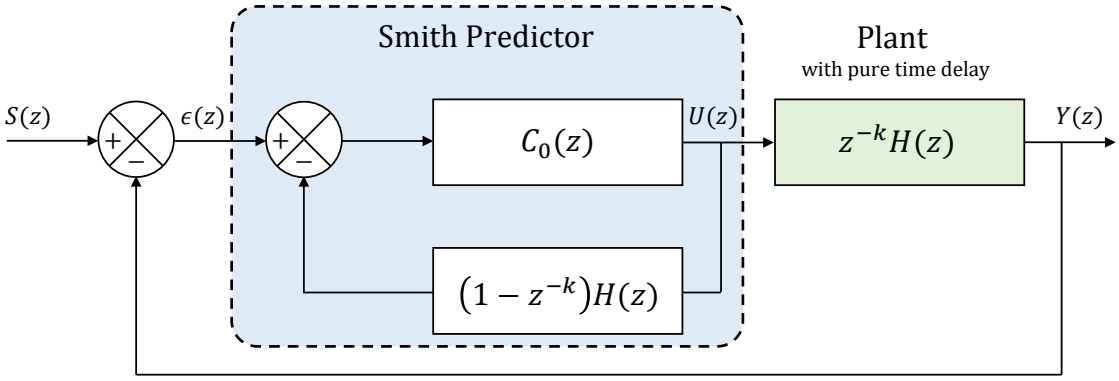


Figure II.10: Smith predictor block diagram.

of the Smith predictor one considers the following system:

$$H(z) = \frac{0.09516}{z - 0.9048}.$$

Assuming it has a pure time delay of $k = 10$ samples. One considers the following PI controller:

$$C(z) = \frac{z - 0.9}{z - 1}.$$

It was designed to get a settling time around 0.3 s without overshoots to a step reference. The results of the PI controller and the PI along with a Smith predictor are presented in Fig. II.11. One can notice large overshoots and oscillations in the step response using the PI controller. The PI along with a Smith Predictor allows to handle the delay. One retrieves the desired response tuned with the PI delayed by k samples.

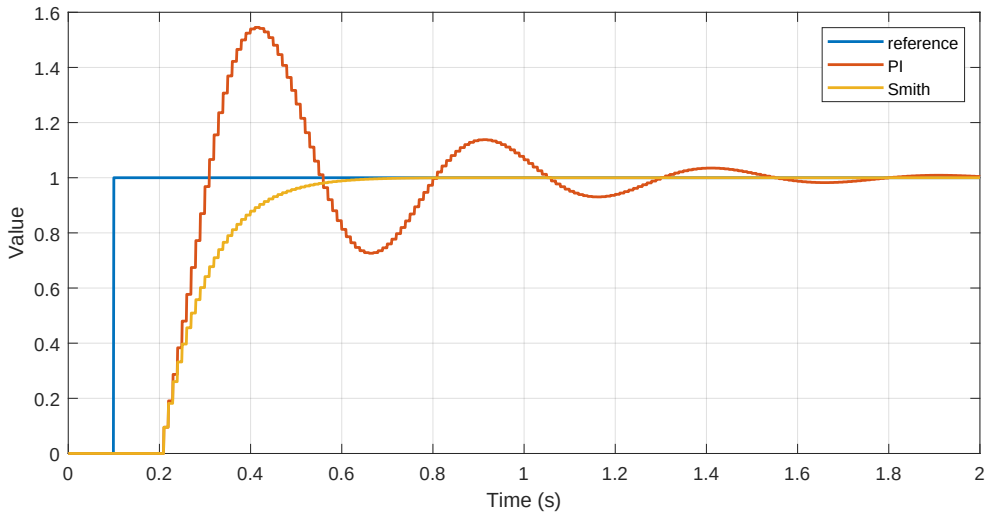


Figure II.11: Example of Smith predictor improvement.

II.6 Data pre-processing

As stated on multiple occasions, data is coming from several sensors. The obtained signals are of varying quality according to the sensor. To perform analysis or learning on it, it might be required to pre-process the data. Most of the time, we obtain usable data as it is. However, it happens that outliers are obtained or the data is extremely noisy to be used.

II.6.1 Dealing with outliers

There are several ways to manage outliers values. As far as we are concerned, we deal with temporal data. We have decided to replace the outlier values by the average of the two surrounding values. For instance, an outlier can appear for one value of acceleration and all other variable of interest can still be correct at this time step. Thus, instead of deleting that outlier and all correct measures, we decided to replace it by that simple approximation. Other methods to deal with outliers exist, but that occasional solution worked for considered problems.

II.6.2 Filtering with Savitsky-Golay

For very noisy curves, here especially data coming from the IMU, it is often necessary to smooth them before using them. We decided to use *Savitsky-Golay* [SG64] algorithm to filter and smooth data. This filter is a method used in signal processing for smoothing a curve and extracting its successive derivatives. The algorithm proposes a smoothing using local least-squares polynomial approximations. Its two main parameters are:

- the windows size l,
- the polynomials order n.

On each considered window of size l, centered at i a polynomial of degree n is determined. The value of the polynomial at this point is the point i of the filtered signal.

Examples of smoothing with Savitsky-Golay filter using IMU acceleration data is shown in Fig. II.12.

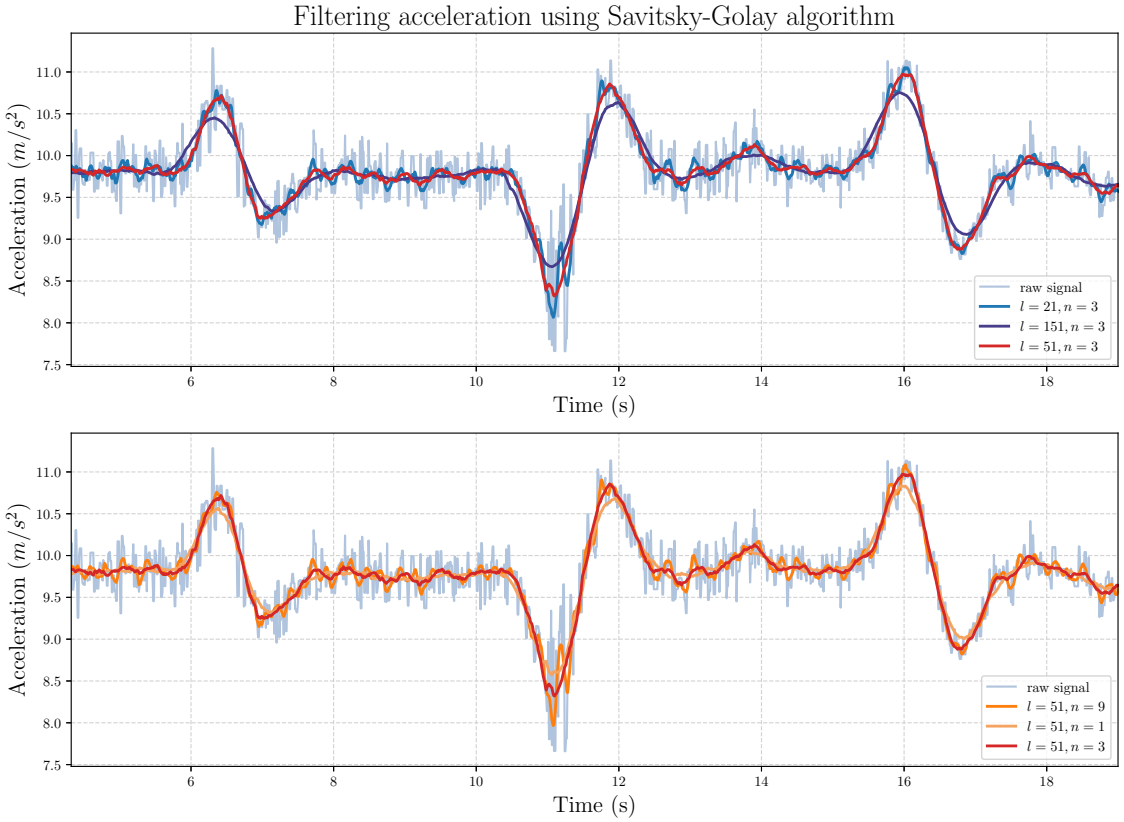


Figure II.12: Example of Savitsky-Golay filtering on acceleration IMU data.

The upper graphic in Fig. II.12 presents the influence of the window size on the filtering. For a fixed polynomial order, the wider the window, the smoother the curve. The bottom graphic in the same figure demonstrates the effect of the polynomial order on the filtering. It is usually unnecessary to use order higher than three.

For experimental purposes, we will use Savitsky-Golay for both smoothing and getting second order derivatives of positions signals. Indeed, the used IMU produces very noisy data, in particular acceleration data. This data is not usable as it is. The use of this filter allows to obtain an estimation of the acceleration using precise position data from the motion capture system. An example of acceleration estimation in z-component using position is presented in Fig. II.13.

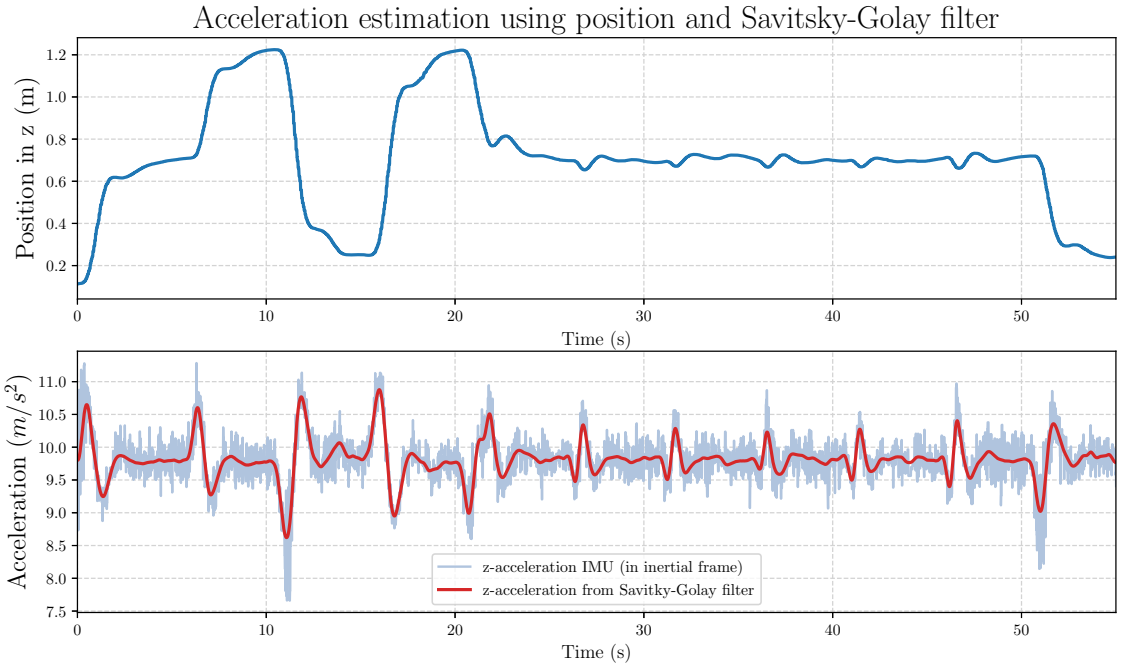


Figure II.13: Example of Savitsky-Golay filtering for getting second order derivative. The first subplot is the z position of the quadrotor obtained from the VICON. It is used to obtained the red curve in the second subplot.

II.7 Conclusion

This chapter focused on the experimental aspect of this thesis, presenting the quadrotor and the experimental platform used. The experimental basis has been established and will be used in the different chapters to come. The details of the control laws using the architecture and the tools presented will be provided in the following. The experimental aspects, concerning the test missions to validate the algorithms will be specified in the corresponding chapters.

Chapter III

Integration of learning into control for linear behavior fitting

This chapter deals with the issue of easy controller design while maintaining expected linear behavior fitting for tracking scenarios. After introducing the problem, the linear controller used for tracking is presented. Using only this standard controller will certainly lead to unexpected behavior of the quadrotor during tracking. We will instigate the derivation of a deep neural network feedforward correction term to correct the flight tracking behavior. To assess the performance of such an enhanced controller, both simulation and experimentation are performed, including step flights, near ground motions and wind disturbance scenarios.

Contents

III.1 Introduction

III.1.1 Motivations

As presented in the general introduction, quadrotor autopilot research has made many advances in recent years. An efficient autopilot for tracking purposes consists of two elements: a fast trajectory generation [RBR16]; [BD18] and an effective controller [BM12]; [FFS18].

On one hand, a fast trajectory generation is a key component to have satisfactory flight performance. It should be capable to face unexpected situations such as moving obstacles and it must be fast enough to be implemented in real-time. [RBR16]; [BD18] propose a polynomial path generation to get fast and feasible trajectories. It is low-cost in terms of computation and complexity. In this manuscript, we will not focus on the trajectory generation part, but it should be remembered that this is an axis of research that needs much attention as well.

On the other hand, the controller must be efficient to follow the proposed trajectory generation. To do so, it must be reactive and capable to deal with all type of disturbances that may occur during flights: for instance wind gust or disturbances generated by ground effect. The complete modeling as presented in chapter I, can be a first step to design efficient controllers. For instance, work of [BM12] presents a more accurate quadrotor model than the commonly used model (I.33). It is next used in a nonlinear model predictive control scheme and demonstrates flight performance improvement. The problem with using such complex models, is that it leads to a more complex tuning (e.g. the gains of the MPC cost function, optimizer parameters, etc.). It also increases the computational complexity as it requires the implementation of non-linear solvers. Such proposed solutions therefore require powerful controllers to be embedded in the device. A possible solution is a trade-off: the modification of nonlinear algorithms solvers, to obtain a sub-optimal but acceptable solution. Another solution is to use a simpler model, which will not require large computational tools. A linear model is easier to handle, popular approaches often consist in linearizing a nonlinear model to avoid complexity [MCO19]. However, obtained tracking performance might not be as expected because simplifications were made.

In parallel to all of that, progress in artificial intelligence is breaking through in all areas. These are now extending into the world of systems, opening up whole new aspects of modeling and control. It started with machine learning techniques that are now combined to classical tools of modeling and control. The survey [Hun+92], presents possible combinations between machine learning techniques and standard control approaches to improve systems performance. Other works focus on hybrid approaches, mixing classical controller helped with intelligent routines [Wan+15]; [Efe11]; [Li+16]; [Kau+19]; [Tor+21]. A good example of such combination, is the use of neural network to tune PID gains [Wan+15] or non-integer $PI^{\lambda}D^{\mu}$ gains [Efe11]. These papers show the effectiveness of such combination. In [Tor+21], Gaussian processes are used to model disturbances which are used to complete the quadrotor dynamical model and it is then used along with a MPC. That solution allows to reduce tracking errors at high velocity. The authors of [Li+16] propose a DNN to determine a reference trajectory. It demonstrates an improvement in trajectory tracking with smaller errors while

using the standard PID controller. Finally, a predictive control for racing quadrotor is presented in [Kau+19]. The vehicle flies through moving gates. The spatial position of the gates is predicted by a convolutional neural network and sent to the control. Using this approach, the team was able to win in half the time of the second team.

Deep learning methods can be also used to model system dynamics, as presented in work of [MW15]. The obtained model can be then used with a well chosen control law such as model predictive control. Other studies [ACN10]; [PA15] demonstrate the interest of learning method for modeling purposes, and especially complex aerodynamics difficult to model. In that particular case, for a helicopter system, the commonly neglected vibration phenomena, so as other complex aerodynamics, are learned with a ReLU network. Encouraged by these results obtained on helicopters, the use of learning methods has progressively been integrated in the modeling of such phenomena for UAVs and quadrotors. [Ban+16] uses shallow neural network that aims at modeling whole quadrotor dynamics. In the more up-to-date research of [Shi+19], deep neural network is used to model the complex ground effect disturbance. They next used a model predictive control, coupled with this DNN to improve the landing behavior of quadrotor.

The challenge of designing an efficient controller that fits a given expected behavior for tracking can also be solved using reinforcement learning methods. RL algorithms for quadrotor control is developing a lot, as evidence of this the following works [Hwa+17]; [Lam+19]; [Pi+20]. Nevertheless, these contributions are unsatisfactory because they require a huge amount of data. Moreover, the collected data must be relevant to correctly learn a controller. It requires an extensive exploration process, often done randomly [SB18], using epsilon-greedy exploration, for instance. Hence, for high-order systems as quadrotors, it necessitates a lot of data and computation time. Secondly, such unsupervised approaches do not provide any stability insurance for the closed-loop system. It can be very problematic for robotic applications.

According to all the above mentioned discussion, one notice two main issues in an efficient autopilot design:

1. First, algorithms presented in research communities are becoming increasingly sophisticated and are often not even implementable outside of the simulation scope. On the other hand, we notice that in the industry the use of PID controllers is still widely done [ACL05] and it is not an exception for UAVs and quadrotors. It is even the case for open-source software such as *PX4* [Noa] or *Ardupilot* [Ard18], that uses cascaded PID controllers for piloting positions, velocities, attitude and angular rates.

2. Using standard controllers, such as PID, is often not enough to perform an efficient tracking under disturbances. Indeed, the quadrotor non-linearities might become very problematic or external disturbances might be difficult to handle. Many of new methods using learning can help improving the trajectory tracking. Especially, using them to model complex aerodynamics that are very often neglected or unknown and that are applied to the quadrotor during flights.

III.1.2 Proposed DNN-based solution

To cope with previously stated issues, we propose a DNN-based controller. It is a mixed solution between a standard cascaded controller and a deep neural network feedforward corrective block. In order to reduce the design complexity and to preserve stability, a linear cascaded control is used, here in it simplest form, a PD feedback control using Linear-Quadratic Regulator (LQR) synthesis. A deep neural network is added to improve the tracking by learning errors. It uses collected data to learn the error between the linear model and the real system dynamics, both controlled with the linear cascaded controller. Fitting an a priori linear behavior is particularly interesting in the case of obstacle avoidance. Indeed, it is easier to predict the error between the reference and the obtain behavior when the system behaves linearly. It also eases a fast trajectory generation. The DNN will not only learn a complex and a nonlinear dynamic error, but will also include parametric errors. The error modeled and identified by the DNN is next introduced in the real-time command so that the quadrotor behavior fits to the linear expected dynamics.

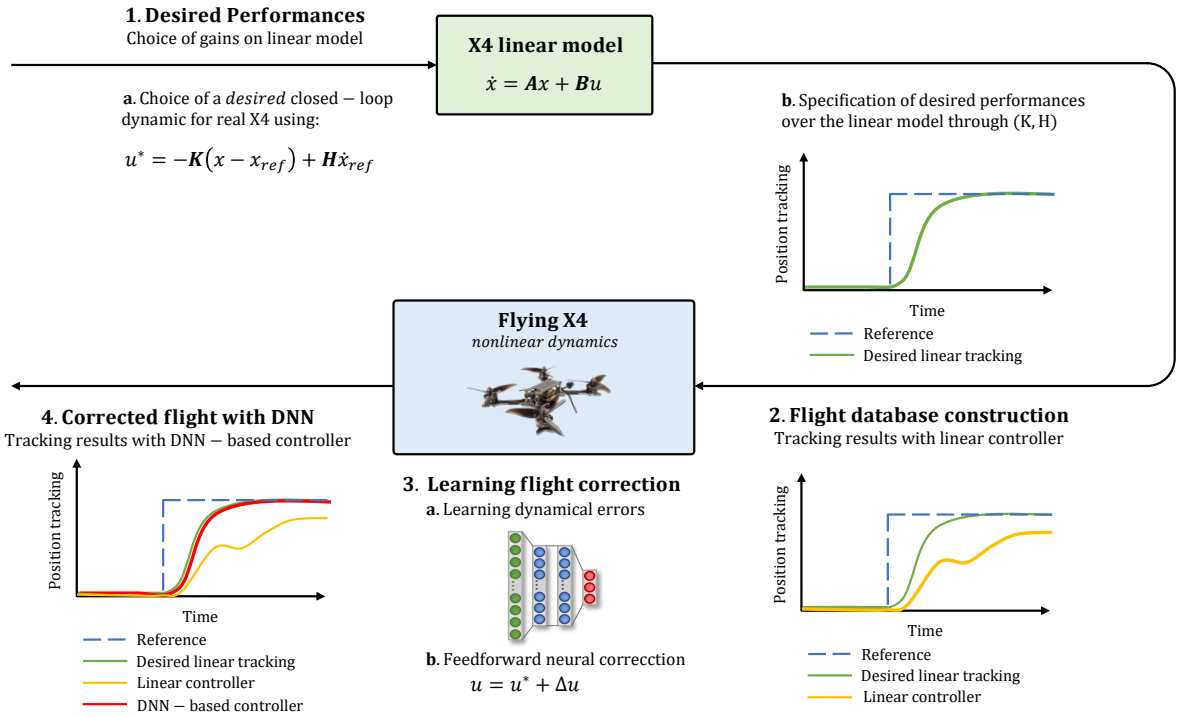


Figure III.1: The proposed methodology for quadrotor linear behavior fitting. It is a four steps approach. Firstly, one designs a linear controller to reach desired linear behavior. Secondly, we create a database of simulation/experimental flights. Thirdly, we learn a DNN network using selected inputs, to predict the feedforward correction to be applied to the position and velocity loop. Fourthly, after learning the neural network offline we can apply the DNN-enhanced scheme.

The proposed methodology is summarized in Fig. III.1. It is composed of four main steps:

1. **Choice of desired performance:** Desired performance is chosen on a linear model (section III.2) of the quadrotor using a cascaded architecture (section III.4). It is based on LQR gains for position, velocity and attitude loops. All gains are tuned to get an expected response with given settling time and limited overshoots for a step in all components.
2. **Database construction:** With the designed controller, the quadrotor is able to track a given trajectory reference in different mission situation. For instance, steps in position, for abrupt changes, or near ground flights. The obtained behavior will not be as expected on the linear model, due to the nonlinear dynamics. Thus, a data collection is made in order to learn errors made during flights.
3. **Learning the flight errors:** The database can now be used to learn errors made during the tracking. To that end, a deep neural network is built and trained (section III.3). Once the learning done, it is integrated in the control via a feed-forward action in the control command (section III.4).
4. **Flight using the learn DNN:** Finally, one can deploy the whole control scheme using the DNN-based correction controller. Flights are performed in different tests cases (section III.5): standard flight, near ground mission and in a windy condition. The newly built controller allows to correct the tracking and fit the expect linear behavior from step n°1.

III.2 Linear quadrotor model and control

As previously stated, a manner to easily setup controllers, is to use linear control tools, for instance using PID controllers. Methods of adjustment without knowledge of the quadrotor exist, but by using a model, even a linear one, one can design a controller that meets the expectations in terms of desired performance: settling time, overshoot etc. without many trials. One can refer to open-loop tests that are difficult to perform on quadrotor system. Thus, using a simple knowledge of quadrotor, here a linear model, can help us setting up a linear controller.

III.2.1 Quadrotor linear model

The linear model, used for control purposes, is obtained using the simplified model given by (I.33) presented in chapter I. We extract from it the equations of interest to perform position/velocity control and angular control. The system of equations to describe the quadrotor is given by:

$$\left\{\begin{array}{rcl} \dot{\xi} & = & v, \\ m\dot{v} & = & -mg\vec{i_3} + \boldsymbol{R}T\vec{b_3}, \\ \dot{\zeta} & = & \boldsymbol{W}(\phi,\theta,\psi)\Omega. \end{array}\right. \tag{III.1}$$

In model (III.1), it should be noted that the third equation was replaced using (I.2). We are directly using Euler angles instead of the rotation matrix formalism. Indeed, our position and velocity controllers will provide reference angles to the attitude controller and use it without conversions. A well known drawback of this representation is its singularity: at $\theta = \pi/2 + k\pi$ for $k \in \mathbf{Z}$. But this will be avoided by carefully designing the trajectory generation and by using angular saturation.

Let us define the state vector composed of positions, velocities and angles, and the command vector composed of thrust and angular velocities. They are denoted as follows:

$$X := \begin{bmatrix} \xi^T, & v^T, & \zeta^T \end{bmatrix}^T,$$

$$U := \begin{bmatrix} T, & \Omega^T \end{bmatrix}^T,$$

with $X \in \mathbf{R}^9$ and $U \in \mathbf{R}^4$. An equilibrium point can be chosen at hover conditions. That is to say at a given position in space, with no speed and zero angles. To simplify the problem, we are assuming the yaw angle is here fixed to zero. Using (III.1), it is clear that at hover only the gravitational force has to be compensated, which can be easily understood physically. The equilibrium point is given by the equilibrium variables X_{eq} and U_{eq} as:

$$X_{eq} := \begin{bmatrix} \xi_{eq}^T, & \mathbb{0}_3, & \mathbb{0}_3 \end{bmatrix}^T,$$

$$U_{eq} := \begin{bmatrix} mg, & \mathbb{0}_3, \end{bmatrix}^T.$$

As commonly done in system analysis and control, one will focus on variation of the quadrotor around the chosen equilibrium. We introduce deviation variables using previously defined equilibrium point, they are given by:

$$\tilde{X} := X - X_{eq},$$

$$\tilde{U} := U - U_{eq}.$$

A linear model of the nonlinear quadrotor system from (III.1) can be obtained at (X_{eq}, U_{eq}). It is given by the linear state space model (III.2):

$$\dot{\tilde{X}} = \boldsymbol{A}\tilde{X} + \boldsymbol{B}\tilde{U}, \tag{III.2}$$

where Jacobian matrices $\boldsymbol{A} \in \mathbf{R}^{9\times9}$ and $\boldsymbol{B} \in \mathbf{R}^{4\times9}$ are defined as follows:

$$\boldsymbol{A} = \frac{\partial f_{X4}}{\partial X}(X,U)\Big|_{(X_{eq},U_{eq})},$$

$$\boldsymbol{B} = \frac{\partial f_{X4}}{\partial U}(X,U)\Big|_{(X_{eq},U_{eq})}.$$

Developing (III.2) using expressions of $\boldsymbol{A}$ and $\boldsymbol{B}$, and based on a first order Taylor expansion,

we get:

$$\dot{\tilde{X}} = \begin{bmatrix} \mathbf{0_3} & \boldsymbol{I_3} & \mathbf{0_3} \\ \mathbf{0_3} & \mathbf{0_3} & \boldsymbol{A_{2,3}} \\ \mathbf{0_3} & \mathbf{0_3} & \mathbf{0_3} \end{bmatrix} \tilde{X} + \begin{bmatrix} \mathbf{0_{34}} \\ \boldsymbol{B_2} \\ \boldsymbol{B_3} \end{bmatrix} \tilde{U}, \tag{III.3}$$

where $\mathbf{0}_3$ refers to the square null matrix of order 3, $\boldsymbol{I_3}$ is the identity matrix of order 3, $\mathbf{0_{34}}$ is a null matrix composed of 3 lines and 4 columns and where $\boldsymbol{A_{2,3}}$, $\boldsymbol{B_2}$ and $\boldsymbol{B_3}$ matrices are expressed as follow:

$$\boldsymbol{A_{2,3}} = \begin{bmatrix} 0 & g & 0 \\ -g & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \boldsymbol{B_2} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ m^{-1} & 0 & 0 & 0 \end{bmatrix} \text{ and } \boldsymbol{B_3} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The system given the state space (III.3) is controllable, the controllability matrix is full rank, $rank(\mathcal{C}) = 9$. It is interesting to note that the state matrix $\boldsymbol{A}$ only depends on the knowledge of g constant and $\boldsymbol{B}$ only depends on the total mass of the quadrotor m. Thus, by knowing these two variables we are able to design a linear controller.

III.2.2 Quadrotor linear cascaded PD controller

The proposed general control scheme for quadrotor trajectory tracking is given in Fig. II.8. One needs to design two linear cascaded controllers: the attitude loop and the position & velocity control loop. The inner-loop, for angular control is first presented and is tuned to be fast enough for the outer-loop. Then, the controller for position and velocity is derived. For both controllers, model previously establish by the state space (III.3) is used.

III.2.2.1 Attitude controller

We are first interested in making Euler angles reach the reference angles. Angular references ζ_{ref} are provided by the position and velocity control. Using the third equation from (III.3), it directly comes that $\dot{\zeta} = \Omega$. The Wronskian matrix (I.3) is equal to the identity according to the assumptions made for linearization: angles are small. Then angular speeds are equal to the derivative of angles. We can directly choose the command input Ω such that:

$$\Omega = -\boldsymbol{K_\zeta} \left(\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} - \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}_{ref} \right). \tag{III.4}$$

Then, using (III.4), the desired closed-loop behavior can be achieved with $\boldsymbol{K_\zeta}$. Where $\boldsymbol{K_\zeta}$ gain can be obtained through a linear quadratic regulation synthesis and $\zeta_{ref} = \begin{bmatrix} \phi, & \theta, & \psi \end{bmatrix}^T_{ref}$ is the attitude target coming from the position and speed controller.

III.2.2.2 Position and velocity controller

Before establishing the linear controller used for position and speed loop, we reformulate equation (III.2) using following notations. Let us define $u_\zeta := [\tilde{\phi},\ \tilde{\theta},\ \tilde{T}]^T$, $\tilde{\xi} := [\tilde{x},\ \tilde{y},\ \tilde{z}]^T$ and $\tilde{v} := [\tilde{v}_x,\ \tilde{v}_y,\ \tilde{v}_z]^T$. Then, the linear state space model (III.3) for position and velocity subsystem is written as:

$$\begin{cases} \dot{\tilde{\xi}} &= \tilde{v}, \\ \dot{\tilde{v}} &= \begin{bmatrix} 0 & g & 0 \\ -g & 0 & 0 \\ 0 & 0 & m^{-1} \end{bmatrix} u_\zeta = \boldsymbol{B_\zeta} u_\zeta. \end{cases} \tag{III.5}$$

Defining error variables $\eta_1 := \tilde{\xi} - \tilde{\xi}_{ref}$ and $\eta_2 := \dot{\eta}_1 := \dot{\tilde{\xi}} - \dot{\tilde{\xi}}_{ref} = \tilde{v} - \tilde{v}_{ref}$, previous equation (III.5) can be rewritten as:

$$\dot{\eta} = \begin{bmatrix} \mathbf{0_3} & \boldsymbol{I_3} \\ \mathbf{0_3} & \mathbf{0_3} \end{bmatrix} \eta + \begin{bmatrix} \mathbf{0_3} \\ \boldsymbol{B_\zeta} \end{bmatrix} (u_\zeta - \boldsymbol{B_\zeta^{-1}} \dot{\tilde{v}}_{ref}), \tag{III.6}$$

with $\overline{\boldsymbol{A}} \in \mathbf{R}^{6\times6}$ and $\overline{\boldsymbol{B}} \in \mathbf{R}^{3\times6}$ matrices defined as:

$$\overline{\boldsymbol{A}} = \begin{bmatrix} \mathbf{0_3} & \boldsymbol{I_3} \\ \mathbf{0_3} & \mathbf{0_3} \end{bmatrix} \text{ and } \overline{\boldsymbol{B}} = \begin{bmatrix} \mathbf{0_3} \\ \boldsymbol{B_\zeta} \end{bmatrix}.$$

Using defined model (III.6), we propose to use again a linear quadratic regulator to reach a desired behavior. The state command vector expression is then given by:

$$u_\zeta^* = -\boldsymbol{K_\eta}\eta + \boldsymbol{B_\zeta^{-1}} \dot{\tilde{v}}_{ref}, \tag{III.7}$$

where $\boldsymbol{K_\eta}$ gain is chosen such that we minimize a quadratic cost expressed as follows:

$$J(u_\zeta) = \int_0^{+\infty} (\eta^T(t)\, \boldsymbol{Q}\, \eta(t) + u_\zeta^T(t)\, \boldsymbol{R}\, u_\zeta(t)) \mathrm{d}t.$$

The weighting matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ are tuned to obtain the desired tracking performance. In practice, these matrices are chosen to obtain a given settling time and a given overshoot on the linear model (III.6). The weights of these matrices are found manually, by using the same principles as the PID tuning explained in the experimental chapter, in section II.5.3.

The additional term found in eq. (III.7), $\boldsymbol{B_\zeta^{-1}} \dot{\tilde{v}}_{ref}$, is a feedforward term to anticipate the reference in acceleration. As references are given in position and velocity, one can either get or compute this term following the chosen trajectory generation algorithm.

It is important to note that the proposed initial controller does not include an integral terms, here. Indeed on the linear model it is theoretically sufficient to reach desired behavior only using proportional and derivatives terms. Obviously, static errors will appear in flight,

but it will be managed with the neural correction term. Here, we derived the minimal controller to be tuned to get a functional controller.

III.2.3 Quadrotor linear control results

The $\mathbf{K}_\eta$ gain is chosen such as the linear system settling time is equal to $1s$ and such that the step response does not present any overshoot on each axis. This obtained behavior of the linear system described by (III.2) using linear command is referred as *expected linear behavior*. Indeed, it corresponds to the perfect response of the quadrotor system if it behaves exactly as its linear model. It is obvious that, when using the same controller, on the real quadrotor system, obtained behavior will not exactly fit this expected linear behavior for several reasons. First, the quadrotor will move away from the equilibrium then its behavior will not be exactly as expected. Second, non-linearities and disturbances not taken into account will increase that difference.

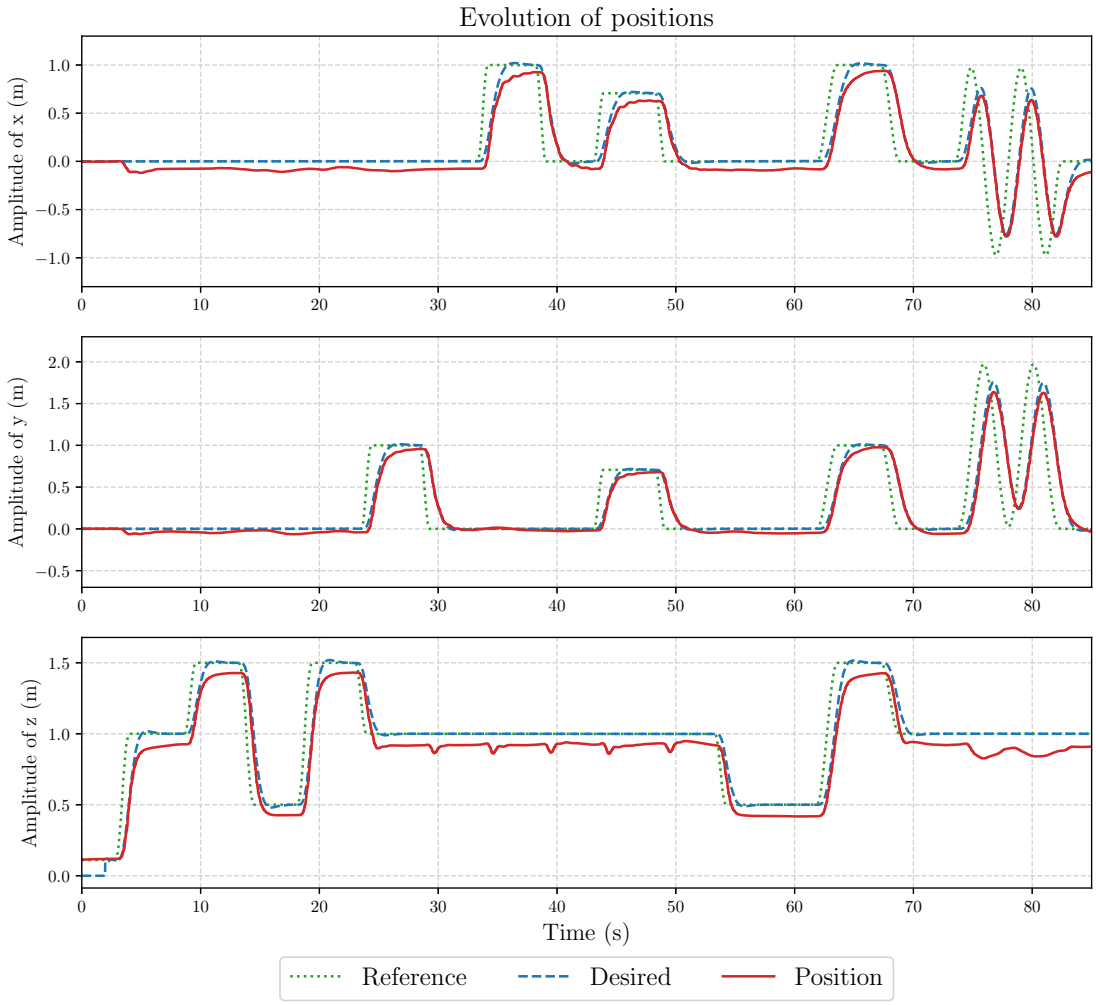


Figure III.2: Tracking results for the linear cascaded controller in a Gazebo simulation test.

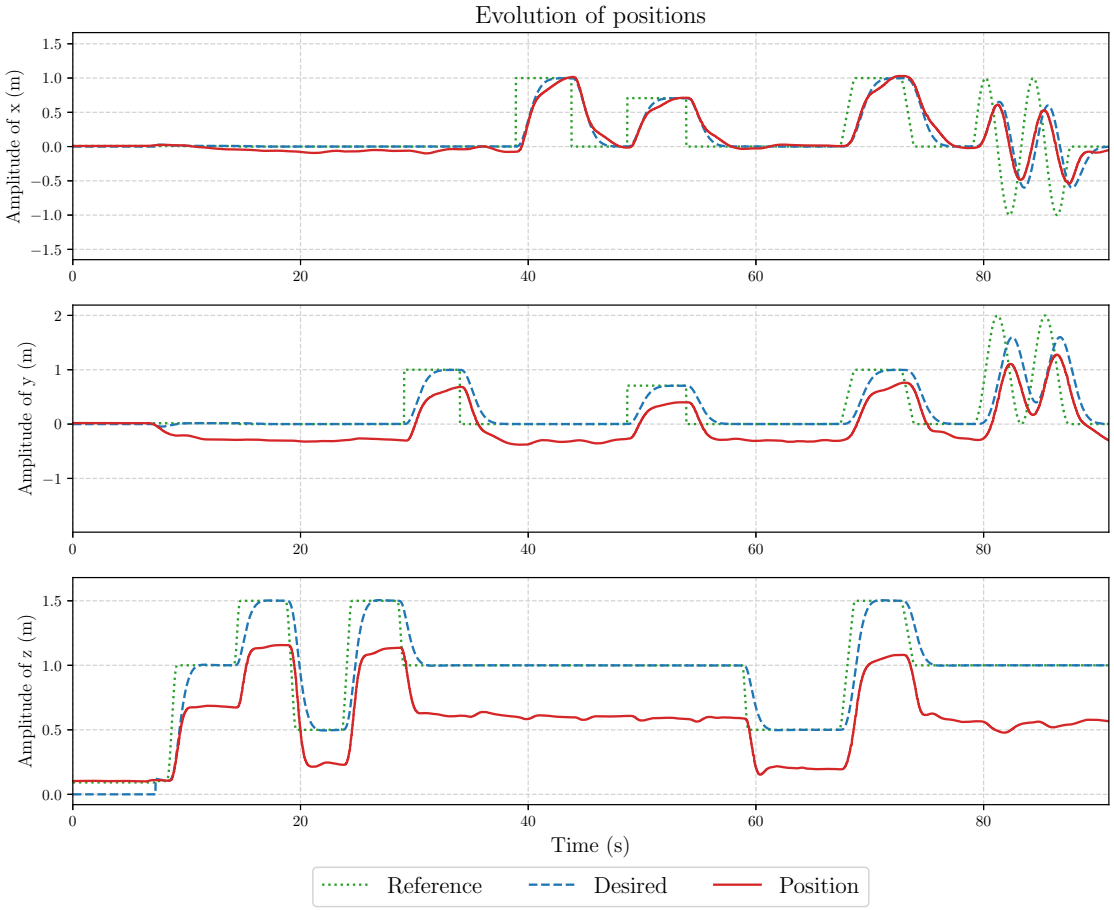


Figure III.3: Tracking results for the linear cascaded controller in a real flight test.

The proposed cascaded linear controller using $\boldsymbol{K_\eta}$ gain is first tested in Gazebo simulation environment. Obtained results are presented in Fig. III.2. The quadrotor is asked to perform steps and a circle in space. Large statics errors can be observed on the position, due to the lack of integrator. It also can be observed that the obtained behavior (red curve) is slower and presents oscillations in comparison to the expected linear behavior (blue dashed curve).

The designed controller is also tested in real flights using the setup presented in chapter II. The results are displayed in Fig. III.3 and confirm the difference between the expected and obtained behavior. Here, static errors are more significant, especially in the z component. On real flight tests, the battery discharge is also greatly affecting altitude over time. It adds a linear decrease in the z-response. It becomes an exponential decrease at the end of battery voltage, but it is not presented here in this scenario.

It is clear that for a concrete scenario such as delivery or other applications, such behavior is not acceptable. The error between expected behavior and real behavior can be explained by the fact that we exclusively use a simple controller, here a linear control, without including disturbance or considering the robustness of the complete system in the control design. The proposed controller must be adapted to deal with these error dynamics in order to get a real

behavior fitting the expected one. The proposed approach is to use a learning-based controller to overcome this issue.

The next section is dedicated to the neural estimation of these error to next compensate them. We will use data to learn as precisely as possible this term. Then, it will be handle in the controller using a feedforward correction term to the linear control.

III.3 Error dynamics neural estimation

We can now formulate a new expression of the complete model (I.31) of the quadrotor using the linearized model. It includes a linear part, from the linear model, and a nonlinear part, representing all the error dynamics and the linearization errors:

$$\dot{X} \quad = \quad \boldsymbol{A}\tilde{X} + \boldsymbol{B}\tilde{U} + \delta_X(X,U), \tag{III.8}$$

where $\delta_X(X,U) = [\mathbb{0}_3,\ \delta_t(X,U)^T,\ \mathbb{0}_3]^T$.

The following of this section presents the proposed structure for the deep neural network used to describe the error dynamics defined by δ_t. It corresponds to the unknown dynamics between the real quadrotor dynamics and the linear acceleration dynamics given by equation (III.8). The deep neural network aims at modeling δ_t, it is denoted by $\hat{\delta}_t$ and is given according to (III.9):

$$\hat{\delta}_t \equiv \dot{v} - (\boldsymbol{A_{2,3}}\tilde{\zeta} + \boldsymbol{B_2}\tilde{U}). \tag{III.9}$$

In the following, we explain the choices made for the DNN architecture and training. Some more general considerations about deep learning are given in the dedicated appendix B. It provides more information on artificial neural network description, training algorithm, etc.

III.3.1 Deep neural network architecture

According to the universal approximation theorem, given in B.4, we can build a feedforward neural network composed of one hidden layer with a finite number of units to approximate a continuous function. Here, we proposed to use a deep neural network to fit the data, it was also proven that multilayer feedforward networks are universal approximators [HSW89]. Based on the nature of that problem, a regression of temporal data, the use of this type of neural networks is well suited [GBC16].

Let first define all weights and bias used to build the DNN: $\boldsymbol{\mathcal{A}_1} \in \mathbf{R}^{N_u,\#\mathrm{x}_t}$, $\boldsymbol{\mathcal{A}_2} \in \mathbf{R}^{N_u,N_u}$, $\boldsymbol{\mathcal{A}_3} \in \mathbf{R}^{N_u,3}$ $\boldsymbol{\mathcal{B}_1} \in \mathbf{R}^{N_u}$, $\boldsymbol{\mathcal{B}_2} \in \mathbf{R}^{N_u}$ and $\boldsymbol{\mathcal{B}_3} \in \mathbf{R}^3$. Where N_u is the number of units, $\#A$ denotes the cardinal of A and x_t denotes the neural network input at time t. The rectified linear unit (ReLU) is used as activation function (h) for all hidden layers. Recalling that

ReLU is defined element-wise by $h(\cdot) = max(\cdot, 0)$.

The expression of the proposed deep neural network is given by eq. (III.10) and an illustration is given in Fig. III.4.

$$\hat{\delta}_t(\mathrm{x}_t) = \boldsymbol{\mathcal{A}_3}^T h(\boldsymbol{\mathcal{A}_2}^T h(\boldsymbol{\mathcal{A}_1}\mathrm{x}_t + \boldsymbol{\mathcal{B}_1}) + \boldsymbol{\mathcal{B}_2}) + \boldsymbol{\mathcal{B}_3} \tag{III.10}$$

This architecture choice, using only fully connected layers, mainly comes from recent promising modeling results from [Ban+16] and [Shi+19]. The number of layers and units was experimentally chosen.

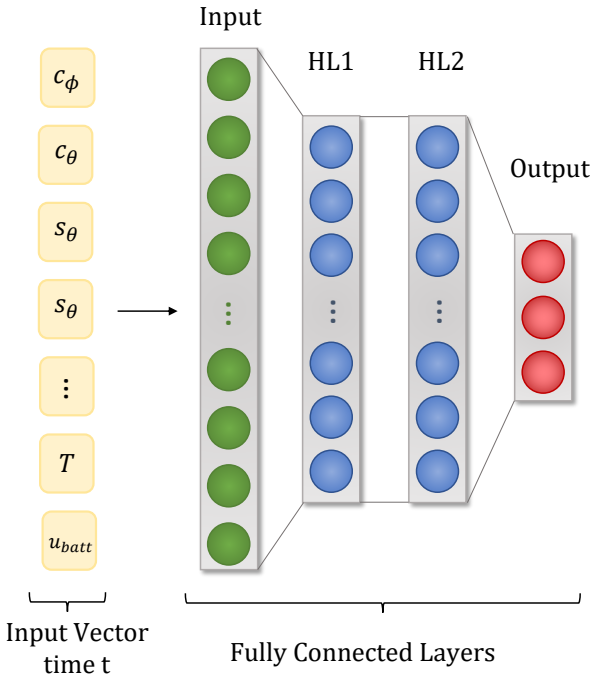


Figure III.4: The deep neural network structure: a feed-forward DNN, composed of N_l hidden layers of N_u units, using ReLU as activation functions.

Concerning the choice of the input vector of the DNN, it is made considering the complete linear acceleration dynamic from (I.31). Both sines and cosines of Euler angles are provided to the input as they appear in (I.1), by developing the rotation matrix. To help the DNN for modeling the drag force, we provide the three spatial linear velocities. The altitude z is provided to help the network to model the ground effect. We do not provide position in x and y as they do not appear in the complete model. Finally, battery voltage tension, u_{batt}, is provided to better learn battery discharge profile. As the network will be train using different batteries, it will learn the several discharge profile type. To summarize, the input state to the DNN, at time t, $\mathrm{x}(t)$, is given by:

$$\mathrm{x}(t) := \begin{bmatrix} c_\phi, & s_\phi, & c_\theta, & s_\theta, & v_x, & v_y, & v_z, & z, & u_{batt}, & T \end{bmatrix}^T \tag{III.11}$$

III.3.2 Deep neural network learning

To learn error accelerations dynamics, we seek to find the weights and bias minimizing the Mean Squared Error (MSE) between predictions and observations. The predictions are expected values obtained from the neural network model $\hat{\delta}_t$ and the observations are the measurements values δ_t^{mes}. Accelerations measurements are collected using the IMU of the quadrotor or using the estimation from Vicon system as explained in the experimental setup description chapter, in section II.6.

Let's define ϑ the optimization variable composed of all weights and biases of the DNN:

$$\vartheta := \{\boldsymbol{\mathcal{A}_1} \quad \boldsymbol{\mathcal{A}_2} \quad \boldsymbol{\mathcal{A}_3} \quad \boldsymbol{\mathcal{B}_1} \quad \boldsymbol{\mathcal{B}_2} \quad \boldsymbol{\mathcal{B}_3}\}.$$

The training of the deep neural network (III.10) aims to solve the optimization problem (III.12) over the training dataset:

$$\min_{\vartheta} \mathcal{L}(\vartheta) = \min_{\vartheta} \frac{1}{N} \sum_{k=1}^{N} \|\delta_t^{mes} - \hat{\delta}_t\|^2, \tag{III.12}$$

where N is the number of elements in the dataset and δ_t^{mes} are error dynamics computed by taking measurements outputs and commands inputs. The objective function given by $\mathcal{L}$ is also referred as the loss function. One aims at minimizing it. This can be done using well-known gradient descent methods using back-propagated gradients optimizers, see B.3.3. For that application, we use Nesterov-Accelerated Adaptive Moment (NADAM) to solve it. This algorithm is explained in appendix B.3.2, algorithm 2. Before making this choice, we tested several algorithms like SGD, Adagrad, Adam, etc. The NADAM optimizer showed good results without spending too much time fine-tuning the learning rate, l_r to get a suitable solution. It also avoids falling in a local minimum, one of the weakness of the SGD algorithm.

All network parameters and hyper-parameters used for training (presented in section III.5) are given in the following Table III.1.

III.4 Neural enhanced controller

The proposed DNN-based controller is composed of the linear controller described in section III.2.2, based on the previously established linear model (III.2), and a feedforward neural estimation term derived from (III.10). The solution is also referred as neural enhanced controller in the following.

Parameter	Value
Number of inputs #x	10
Number of hidden layers n_l	2
Number of units n_u	64
Collection time (s)	30
Batch	256
Epoch	800
Learning rate l_r	0.002
μ	0.9
ν	0.9999

* see NADAM algorithm 2 for more precision.

Table III.1: Parameters selected for learning and for the architecture of the deep neural network.

III.4.1 DNN-based correction controller

According to the equation (III.8), the real system dynamics can be modeled by adding an error term to previously given linear model (III.6):

$$\dot{\eta} = \overline{\boldsymbol{A}}\eta + \overline{\boldsymbol{B}}(u_\zeta - \boldsymbol{B_\zeta^{-1}}\dot{\tilde{v}}_{ref} + \boldsymbol{B_\zeta^{-1}}\delta_t). \tag{III.13}$$

The acceleration error dynamics term δ_t is mainly coming from disturbances and linearization and can be handle with the command input u_ζ to achieve expected linear behavior from linear controller.

The proposed neural enhanced controller is designed as follows:

$$u_\zeta = u_\zeta^* + \Delta u_\zeta, \tag{III.14}$$

where u_ζ^* is the linear quadratic control (III.7) and Δu_ζ is the feedforward correction term based on the deep neural network estimation of error dynamics. The closed-loop form of the system described by (III.13) using (III.14) is given by:

$$\dot{\eta} \quad = \quad \underbrace{(\overline{\boldsymbol{A}} - \overline{\boldsymbol{B}}\boldsymbol{K_\eta})\eta}_{\text{Closed-loop linear behavior}} \quad + \quad \underbrace{\overline{\boldsymbol{B}}(\Delta u_\zeta + \boldsymbol{B_\zeta^{-1}}\delta_t).}_{\text{Term to minimize}} \tag{III.15}$$

According to equation (III.15), the quadrotor closed-loop behavior is the sum of two dynamic terms. The first one describes the desired closed-loop linear behavior. This behavior is fully controlled by the proposed linear controller. The second term, composed of error dynamics, needs to be canceled or at least minimized to get the system response closer to the expected linear behavior. This term cancellation outlines a classical minimization problem stated as follows:

$$\min_{\Delta u_\zeta} \|f(\delta_u)\|^2 = \min_{\Delta u_\zeta} \left\|\Delta u_\zeta + \boldsymbol{B_\zeta^{-1}}\delta_t(\Delta u_\zeta)\right\|^2 \tag{III.16}$$

It is possible to solve it by replacing the error dynamics described by the neural approximation $\hat{\delta}_t$ in (III.16). The following problem has to be solved:

$$\min_{\Delta u_\zeta} \left\| \Delta u_\zeta + \boldsymbol{B_\zeta^{-1}} \hat{\delta}_t(\Delta u_\zeta) \right\|^2 . \tag{III.17}$$

The neural estimation $\hat{\delta}_t$ is a function of Δu_ζ, the correction. It is depending on $\hat{\delta}_t$ according to (III.17), thus it is a nonlinear minimization problem. Such problem can be solved using nonlinear programming. However, it will probably be time consuming to find a suitable solution and will not be implementable for real-time application. To deal with it, a proposed approach is to replace u_ζ by u_ζ^*, assuming that the correction to be made is small in comparison to the command: $u_\zeta \ll u_\zeta^*$. By removing the correction term from the DNN input, we can directly deduce the solution:

$$\Delta u_\zeta = -\boldsymbol{B_\zeta^{-1}} \hat{\delta}_t(u_\zeta^*). \tag{III.18}$$

The knowledge of $\boldsymbol{B_\zeta^{-1}}$ matrix, allows to deduce the correction (III.18) to be applied to the thrust reference T_{ref} and to the two angular reference positions ϕ_{ref} and θ_{ref} to get closer to the linear behavior. The complete control architecture is summarized in Fig. III.5. The correction proposed is only done in the position and velocity loop.

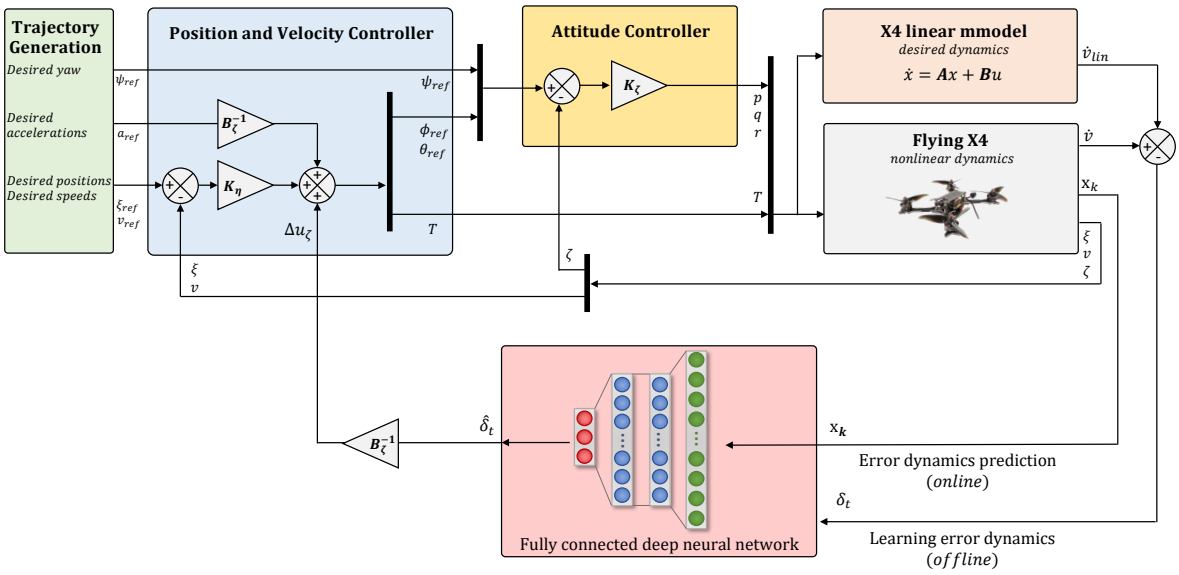


Figure III.5: DNN-enhanced linear controller architecture

III.4.2 Stability analysis

This section focuses on the analysis of the stability of closed-loop system, using the previously established neural controller. First, we consider a candidate linear Lyapunov function given by (III.19):

$$\mathcal{V}(t) = \frac{1}{2} \|\eta(t)\|^2 . \tag{III.19}$$

Next, we define the deep neural network prediction error as $e_\eta = \delta_t - \hat{\delta}_t$. Introducing the minimum eigenvalue of the closed-loop linear system matrix: $\lambda = \lambda_{min}(\overline{\boldsymbol{A}} - \overline{\boldsymbol{B}}\boldsymbol{K_\eta})$, it can be written that:

$$\dot{\mathcal{V}}(t) = \eta^T(\overline{\boldsymbol{A}} - \overline{\boldsymbol{B}}\boldsymbol{K_\eta})\eta + \eta^T e_\eta, \tag{III.20}$$

Considering now inequality, by using the norm operator on (III.20):

$$\dot{\mathcal{V}}(t) \leq \lambda\eta^T\eta + \|\eta\| \cdot \|e_\eta\|. \tag{III.21}$$

Using the definition of $\mathcal{V}$ in (III.21), one obtains:

$$\dot{\mathcal{V}}(t) \leq 2\lambda\mathcal{V} + \sqrt{2V}\,\|e_\eta\|. \tag{III.22}$$

We now introduce a new variable, being the square root of the previously established Lyapunov function candidate, $\mathcal{W} = \sqrt{\mathcal{V}}$, a time derivation of $\mathcal{W}$ gives:

$$\dot{\mathcal{W}} = \frac{\dot{\mathcal{V}}}{2\sqrt{\mathcal{V}}}. \tag{III.23}$$

Making the assumption that $\|e_\eta\|$ is bounded by a constant given by ε_η and using eq. (III.23), the inequality in (III.22) becomes:

$$\dot{\mathcal{W}} \leq \lambda\mathcal{W} + \frac{\varepsilon_\eta}{\sqrt{2}}. \tag{III.24}$$

Given the initial condition on $\mathcal{W}(0) = \frac{\eta_0}{\sqrt{2}}$, the solution to (III.24) is established by (III.25):

$$\mathcal{W}(t) \leq \frac{1}{\sqrt{2}}\left(\frac{\varepsilon_\eta}{\lambda} + \|\eta_0\|\right)e^{\lambda(t-t_0)} - \frac{\varepsilon_\eta}{\sqrt{2}\lambda}. \tag{III.25}$$

The tracking error η can be then bounded by:

$$\|\eta(t)\| \leq \|\eta_0\|\,e^{\lambda(t-t_0)} + \frac{\varepsilon_\eta}{\lambda}\left(e^{\lambda(t-t_0)} - 1\right). \tag{III.26}$$

As a result, the tracking error exponentially converges to a ball, centered at the initial η_0 and with a radius of $\frac{\varepsilon_\eta}{\lambda}$, according to (III.26). One notices that the radius is directly proportional to the prediction error made by the deep neural network and inversely proportional to the eigenvalues of the closed-loop linear system. It must be noticed that the assumption stating a bound on the absolute value of the prediction error is valid for values in the database used for training. However, experimentally during flight, one may encounter values not present in the database. It is then needed to have a sufficiently large enough database which is close enough to all desired flights to unsure stability of the quadrotor with this neural controller. The following chapter IV proposes a solution to better handle the closed-loop stability during flights.

III.5 Simulation and experimental results

In this section, training and validation phases of the deep neural network are first explained. Previously proposed controllers, the linear controller and the DNN-enhanced controller, are then tested in the ROS/Gazebo simulation environment and in experimental flights using the motion capture room described in section II.3.

III.5.1 Training and validation database

To obtain the proposed deep neural enhanced controller, it is first needed to train the DNN presented in equation (III.10). The training process is done using a flight database, also called training dataset. The database $(\mathbf{x}^{\boldsymbol{mes}}, \boldsymbol{\delta_t^{mes}})$ is composed of measured input values and linear acceleration errors δ_t^{mes} computed between measured accelerations and estimated expected linear behavior. As explained, in section II.4.2, measured values are necessary to create the database. They are stored using a personalized ROS message. Values of interest are then selected to create $(\mathbf{x}^{\boldsymbol{mes}}, \boldsymbol{\delta_t^{mes}})$. Two different neural networks are trained: one for simulations and one for experimental tests, each one is trained with its respective dataset.

For both simulation and experimental tests, the database is composed of data collected using the initial cascaded linear control architecture presented in section III.2.2 or equivalently, using the DNN-enhanced controller by setting the DNN contribution to zero, $\Delta u_\zeta = \mathbb{0}_3^T$. It is composed of numerous basic movements in the 3D space, restricted to the flight zone area and capabilities of the initial controller (i.e: no acrobatic movements are performed nor included in database). The training dataset is including translations in all planes (x, y, z, individual and combined), steps in all individual directions and coupled steps. It also includes rotations with circles, spiral motions or lemniscate motions. Those motions are done at different altitude, including being close to the ground to learn the ground effect (see section I.4.1, for more information on that effect). It must be noted that for simulation database construction, the ground effect is not modeled through the environment used in Gazebo, hence the neural network does not learn that effect in this particular case. Some examples of training displacement are shown in Fig. III.6.

The idea behind building a complete database, is to cover a maximum the possible actions that the quadrotor can perform, to better learn the whole dynamic of the device, and thus better predict and correct the errors done during flights. Finally, we restrict the velocity of the generated references up to $3m/s$, and cover several speed in between.

We propose to cover more an hour of flight, with the previously described motions. It is sufficient enough to cover all that motions, without including redundant data in order to correctly and quickly learn the deep neural network. As stated in section III.3.2, NADAM algorithm is used to train the DNN using hyper-parameters in Table V.1. In addition, during training procedure elements are shuffled, to get as much as possible independent data. The learning rate l_r is decreased by means of a multiplication factor of $\frac{2}{3}$ when there is no improvement on the loss function after a given number of epochs. Finally, one-third of the

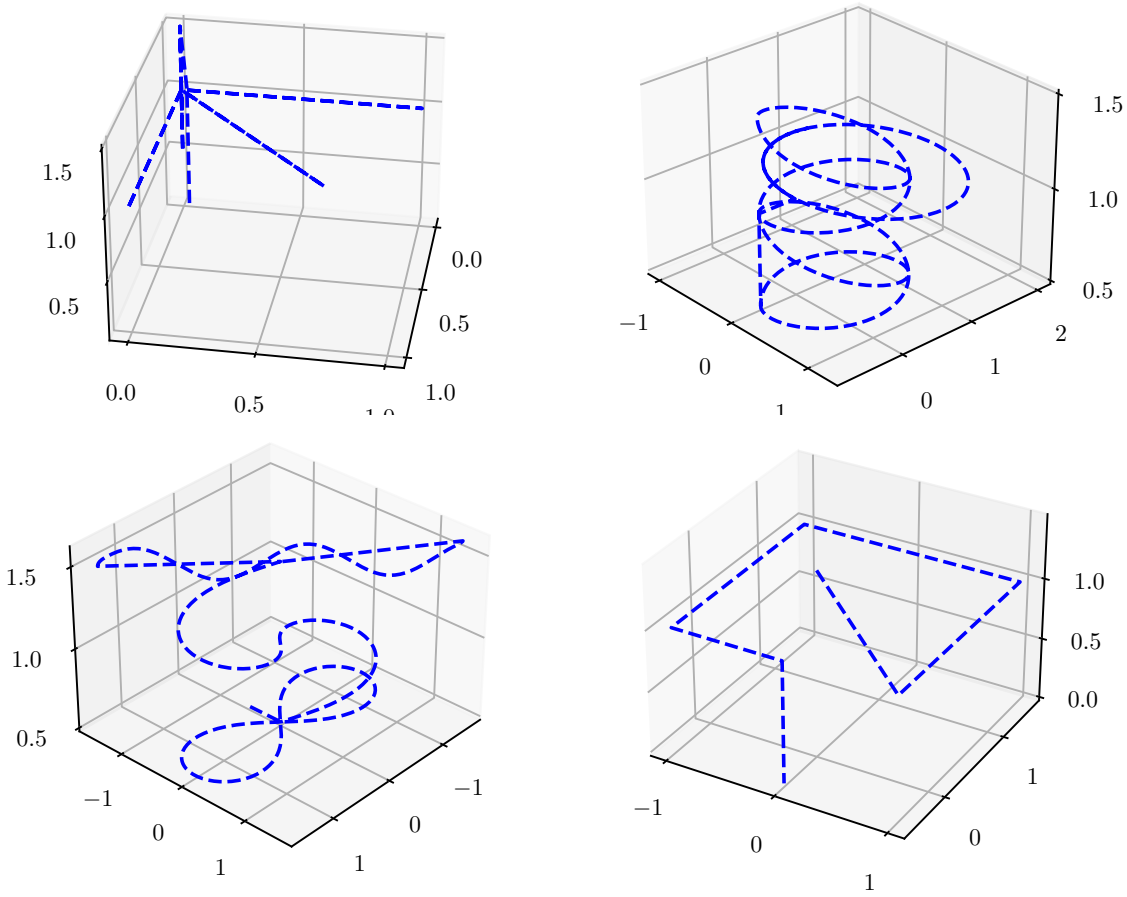


Figure III.6: Example of trajectories: step, circles, spirals, etc. that are included in the database. Distances are in meters.

created database is used as *validation dataset*. That set is used to validate the training and to help making a choice of a deep neural network. It must be sufficiently correctly learned and it must not overfit to the training data.

All those parameters choices were done empirically in order to get a satisfying compromise between speed and convergence of the algorithm. A deeper study should be carried out to get a more optimal compromise. But for this application, choices and results were sufficiently convincing to assess the effectiveness of the proposed methodology.

The training is performed on a Intel® Core™ i7-8665U CPU and is taking around 20 minutes to perform all epochs.

III.5.2 Gazebo simulations results

In this section, we present results of the cascaded linear controller and its DNN-enhanced version in simulation. Indeed, before testing the proposed approach in experimental tests, using the experimental platform, we must validate it and test the DNN-enhanced controller in simulation environment. To get result close to the real-world tests, we performed it on Gazebo simulation environment.

As it is well known in the deep learning community, a third dataset is used to test and assess the choice of the selected deep neural network: the *test dataset*. We thus used an unseen scenario to test the DNN-based controller. This scenario is neither part of the training set nor part of the validation set, see Fig. III.7. It is composed of trajectory in a random selected order at a random velocity speed for trajectory generation.

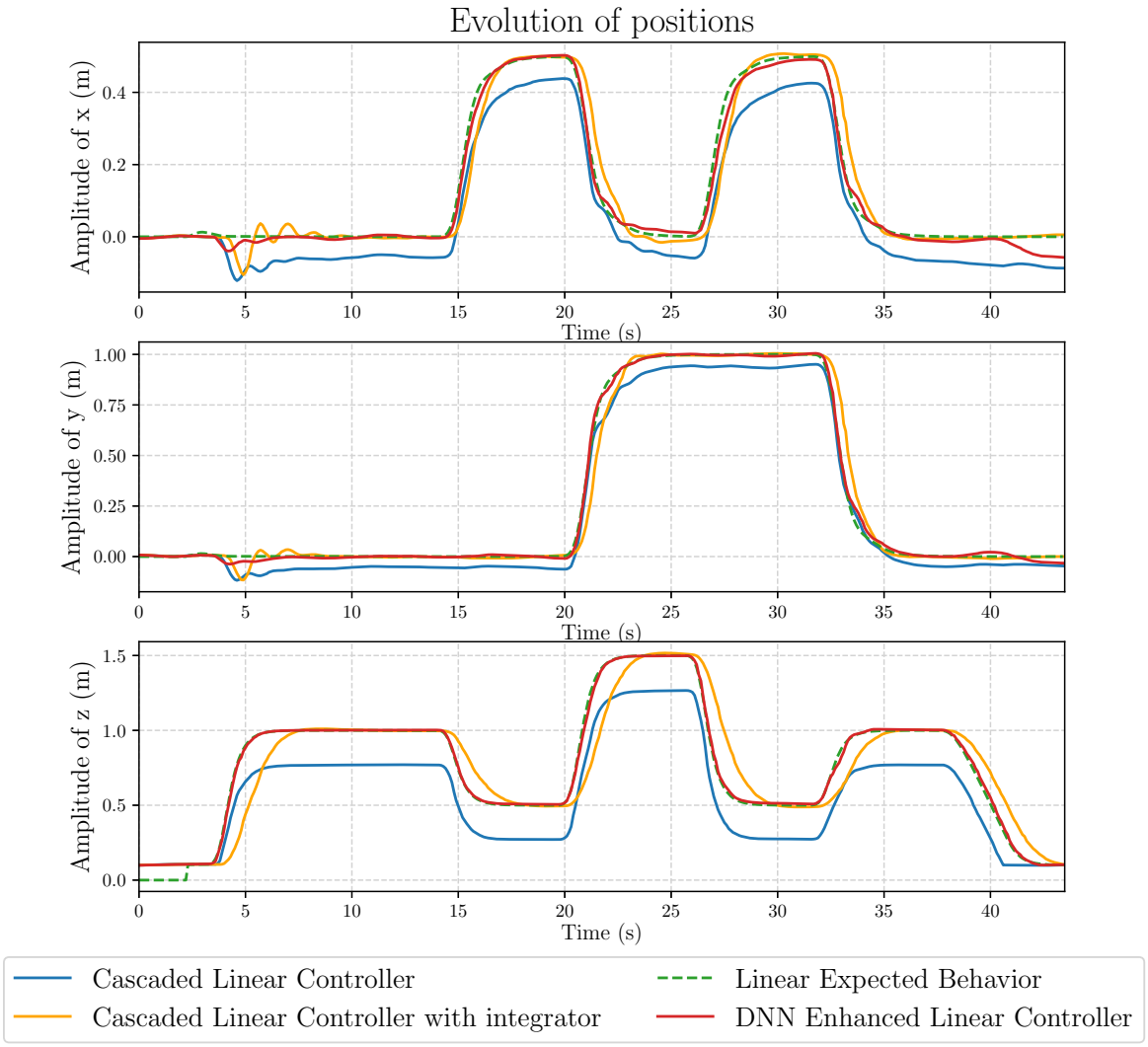


Figure III.7: Evolution of the quadrotor spatial positions for the three controllers in ROS/Gazebo simulation environment, for the test case scenario.

For comparison purposes, a third controller, a cascaded linear controller with an integral action, is implemented and also tested. In order to be correctly compared with the linear cascaded controller and with the DNN-enhanced cascaded controller, the gains of this additional controller with integral action are chosen so that the closed-loop behavior approaches the ideal behavior without integrator. That is to say, to get the expected perfect behavior of the LQR controller on the linear model. Thus, the proportional, integral and derivative gains are selected so that the mean squared error between the integral LQR and the non-integral LQR on the linear model is the lowest. To obtain these gains an optimization must be performed. Here a brute force approach is used but requires several hours of computations.

The results of the three stated controllers are presented in Fig. III.7. The expected linear behavior is represented in a green dashed curve. For the cascaded linear controller, represented with a continuous blue line in the figure, the same remark as in the analysis section III.2.3 can be done. It reveals very high tracking errors and slower response time compared to the linear expected behavior. Again, here the gazebo model does not include battery discharge model, and its impact is thus not visible. The tracking errors are obviously explained be the lack of integrator in the position loop. Moreover the slow response time is due to the linear controller on the nonlinear model. On that test case scenario, tracking errors with the linear cascaded controller are around 7cm on x and y axis and for z axes it is up to 25cm. As previously stated, we expected none of those undesired effects if the quadrotor behaves exactly as its linear model. Those undesired errors, due to nonlinear dynamics of the quadrotor, aerodynamic effects, coupling and unmodeled actuator dynamics are mostly overcome using the proposed DNN-enhanced linear cascaded controller, represented with a red continuous line in the Fig. III.7.

	Baseline linear controller (without integrator)	Baseline linear controller (with integrator)	DNN-enhanced controller
MSE x-axis	0.0049 m^2	0.0021 m^2	0.00029 m^2
MSE y-axis	0.0032 m^2	0.0030 m^2	0.00022 m^2
MSE z-axis	0.0477 m^2	0.0259 m^2	0.00073 m^2

Table III.2: Mean squared errors on each axis, between the expected linear behavior and the response of the three presented controllers, in simulation environment.

The simulation demonstrates that the quadrotor behavior, using the DNN-enhanced controller, is much more closer to the expected linear one, compared to the two other controllers. The deep neural network term has significantly decreased tracking errors. It also has sped-up the response of the quadrotor to better fit the expected response time. The integral controller, tuned to be as close as possible to the linear quadratic controller, also corrects the behavior, by removing the static error but presents a slower response time compared to the DNN version. To better quantify the improvement, MSE on this test trajectory are computed between each controller responses and the linear expected one. All computed MSE are summarized in Table III.2 for each axis. The obtained results confirm that the DNN-enhanced controller is the solution that fits the best the expected linear behavior on all components.

III.5.3 Experimental results in MOCA room

Now that we are ensured that the simulations on Gazebo environment are performing correctly and as expected, one can test and validate the approach on the experimental platform. We recall that the complete setup was described in chapter II. To experimentally assess the effectiveness of the DNN-enhanced controller, we performed three different test case flight: a classical steps scenario in sub-section III.5.3.1, such as in simulation, a near-ground test scenario in sub-section III.5.3.2 and a windy scenario in sub-section III.5.3.3.

III.5.3.1 Steps flight scenario

First, as stated in section III.5.1, we created a database of flights using the quadrotor and the initial linear cascaded controller. In comparison to simulation, the battery discharge is now greatly modifying the final behavior. Thus, we decided to collected data over several scenarios using different batteries. We included several batteries to better cover their discharge profiles, as for the old battery, the battery's discharge is completely different.

After the DNN training, we test all three controllers on an unseen scenario, not presented in the database, see Fig. III.8. To better see the impact of the battery discharge profile, on all controllers, we start each time with hall-full capacity battery and with the same battery for all tests. Obtained experimental results are shown in Fig. III.8.

The blue continuous line represents the linear cascaded controller without correction nor integrator. As for simulation tests, the behavior using that controller presents tracking errors and slower response time. During the flight test, the cascaded linear controller has difficulty to handle the impact of the battery discharge. That discharge is directly impacting the tracking error, especially on z axis. At the beginning of the scenario, the tracking error is nearly equal to 30% and it is increased up to 35% at the end. Finally, undesired oscillating behavior on y axis can be noted. It is particularly visible after $t = 50s$. It is due to the motion coupling, not correctly handle with the linear controller.

The linear cascaded controller including an integral action is represented in yellow in the same figure. It performs better but does not well fit the expected linear behavior. As for the DNN-enhanced controller, in red line, it also completely removed tracking errors. It has learn the battery's discharge profile, thanks to the input of u_{batt} provided to the deep neural network. It is perfectly able to correct the thrust necessary to get to the desired reference. It has also increased the quadrotor velocity on the test scenario and removed undesired oscillations on y axis.

Thus, the quadrotor experimental behavior using the deep neural enhanced controller is much more closer to the expected desired linear one. As for simulation results, the proposed controller is faster and closer to the linear expected behavior compared to the integral controller. Mean squared error are also computed for this experimental test trajectory. They are presented in Table III.3. It confirms the performance of the proposed DNN-based controller.

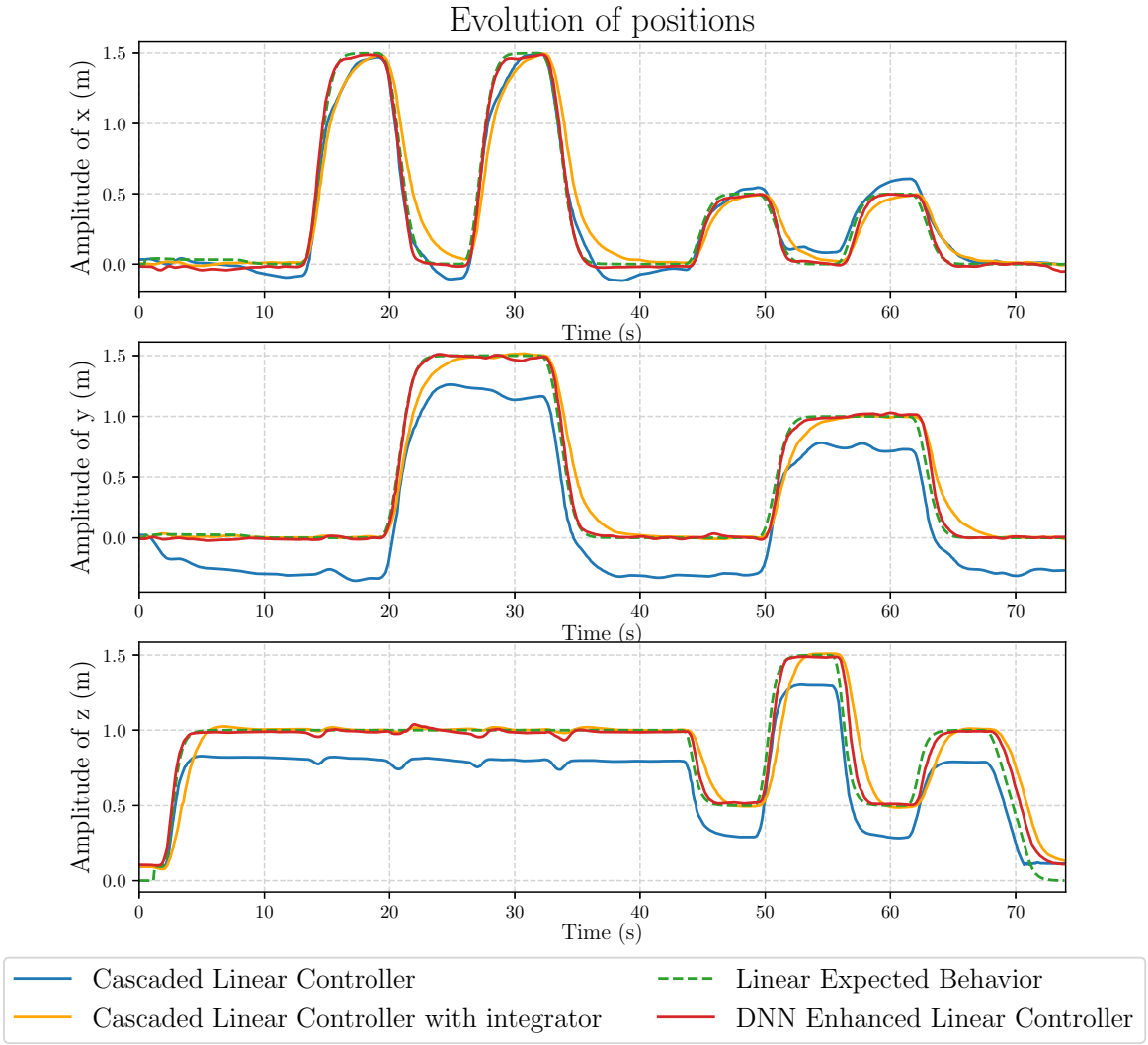


Figure III.8: Evolution of the quadrotor positions for each controller, in real flight test using our motion capture room.

	Baseline linear controller (without integrator)	Baseline linear controller (with integrator)	DNN-enhanced controller
MSE x-axis	0.0060 m^2	0.0150 m^2	0.0016 m^2
MSE y-axis	0.0815 m^2	0.0125 m^2	0.0017 m^2
MSE z-axis	0.0380 m^2	0.0198 m^2	0.0046 m^2

Table III.3: Mean squared error on each axis, between the expected linear behavior and response of the three presented controllers, for the experimental steps flight scenario.

A video of the experimental results is available via the following link: `youtu.be/c70nlsMVi9M`

III.5.3.2 Near-ground flight scenario

To assess improvements of the DNN-enhanced linear cascaded controller with respect to external disturbances, we tested it in a near-ground flight, see Fig. III.10. The quadrotor is asked to track a lemniscate at 15cm from the ground. At this altitude, ground effect is strongly impacting the flight behavior. It is difficult to handle that effect using only the linear cascaded controller. Without integral action, it is very difficult to perform the task, as the battery discharge and the high tracking error in z, may prevent the quadrotor from take-off.

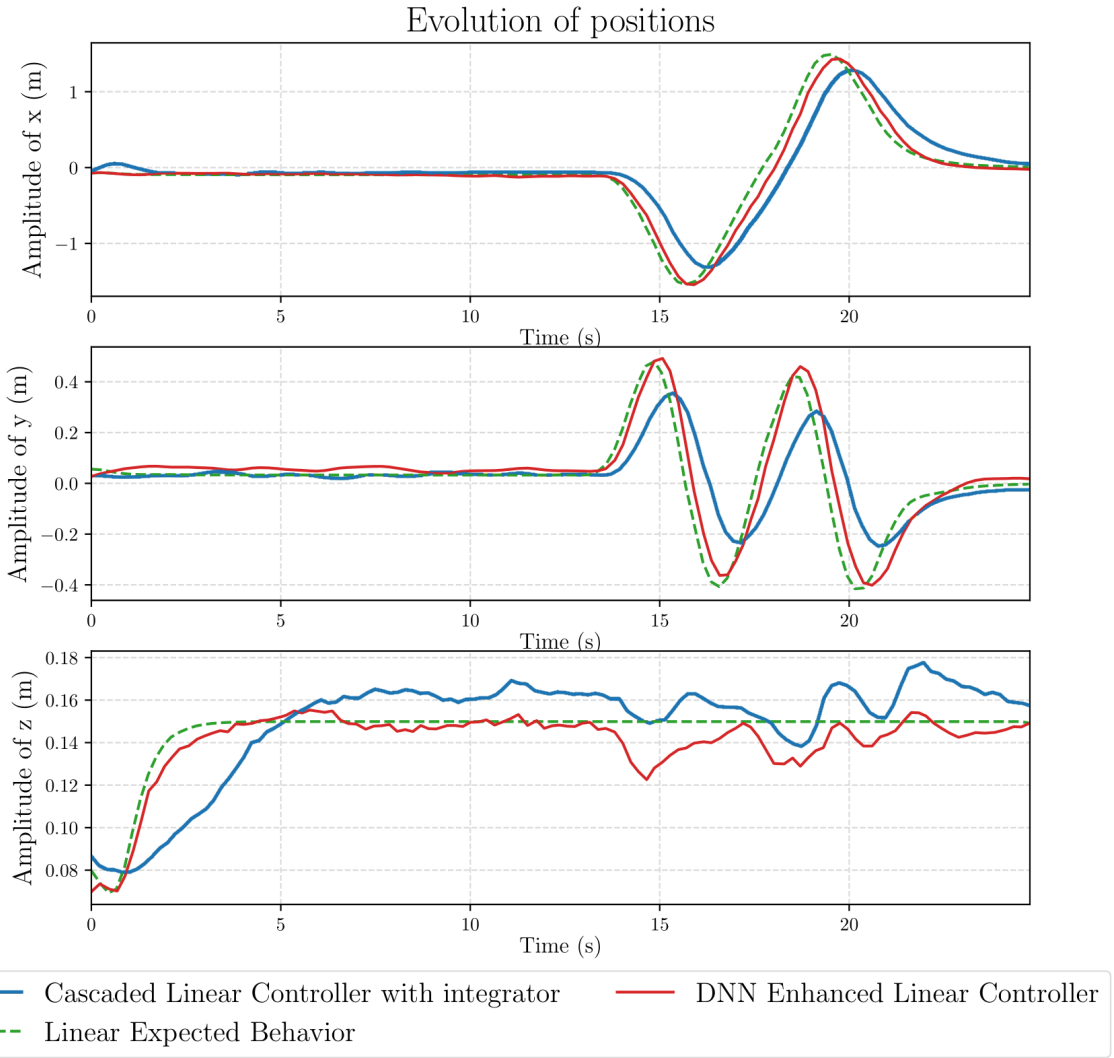


Figure III.9: Evolution of quadrotor positions for the linear cascaded controller versus the DNN-enhanced linear controller in a near -ground scenario.

The same deep neural network is used for testing the DNN-based controller. Indeed, it

was trained on many different scenarios including near ground flights. We present in Fig. III.9 the results of that experiment for the cascaded linear controller with integrator and for the DNN-enhanced linear controller.

As expected, the neural controller is able to correct the behavior and gets closer to the linear expected behavior in that highly disturbed situation. Here, the integral controller is not well rejecting ground effect disturbance, as ground effect is not a constant disturbance. The neural term is able to learn those dynamical disturbances. The proposed DNN-enhanced controller performs better than the integrator in all three axes.

Figure III.10: A picture of the quadrotor flying close to the ground.

III.5.3.3 Wind rejection

Finally, some tests were performed to observe the effectiveness of the proposed DNN-based controller under windy condition. To achieve it, a complement of a database was collected. The approach focuses on a front wind to the quadrotor. It is asked to perform back and forth in front of a constant wind. First, we collected additional data by doing several flights using different wind intensity. The previously established database has then been extended with those tests.

Another deep neural network is learned using this new database, with the same hyper-parameters presented in Table III.1. Then, we applied it to a test case scenario with an unseen intermediate wind gust.

The results are presented in Fig. III.11. Without, correction one remarks that the wind pushes the quadrotor away from it setpoint. In addition, the wind create an additional lift that pushes the quadrotor over its altitude reference. Concerning, the DNN-enhanced controller one notices that the wind disturbance is correctly rejected in these components. It confirms that the DNN-based controller is able to overcome external disturbances.

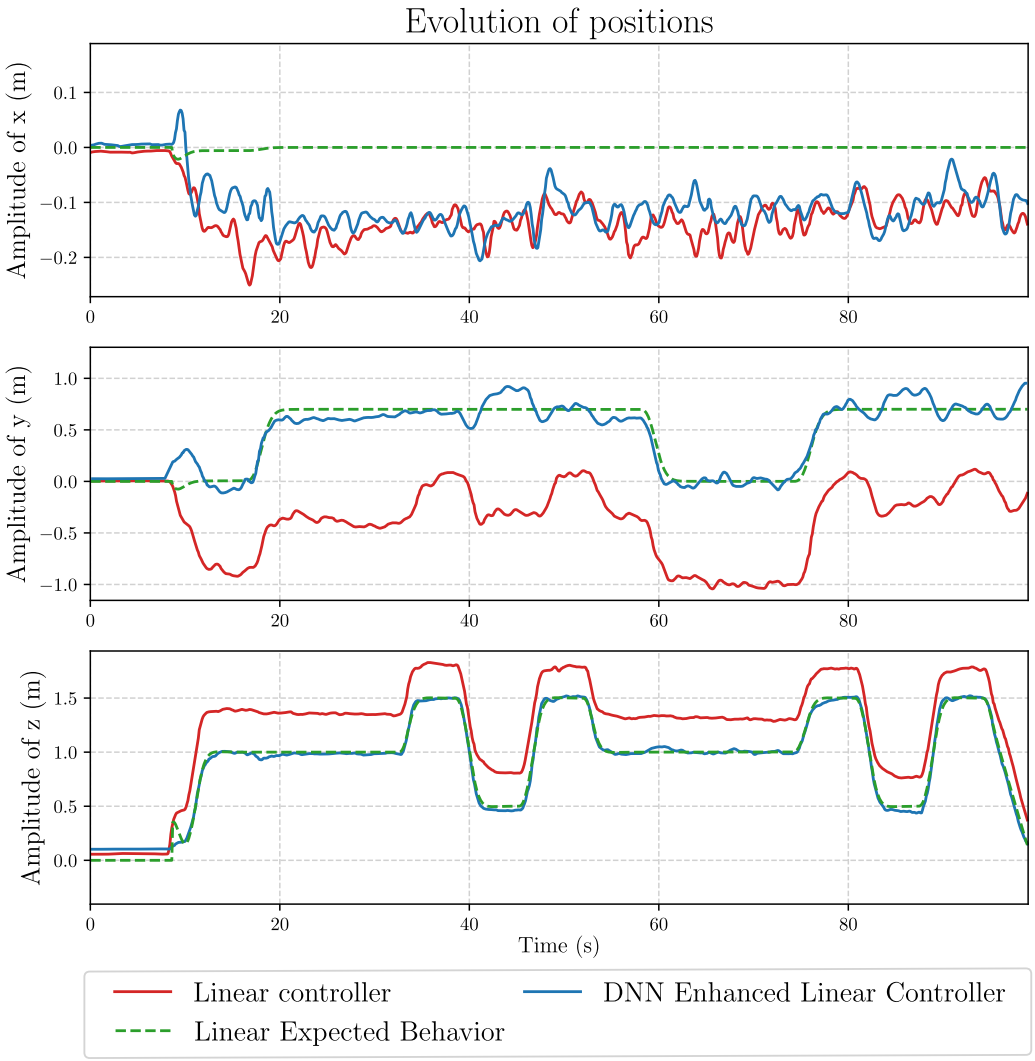


Figure III.11: Evolution of positions under a constant wind.

Those results are preliminary results, a deeper comparison between controllers including observer will be made in the following chapter.

III.6 Conclusion

Let us recall the initial problem, which was the development of an efficient and easy to setup controller. The solution proposed by the DNN-enhanced cascaded controller, largely fulfills its requirements. Beforehand, it is based on the commonly used cascaded control architecture used in industrial and open-source quadrotors. To simplify the implementation and the tuning of the controller, it is based on a linear cascaded controller. The gains are tuned on a linear model. This reduces the problem to a choice of performance criterion on an ideal linear model of the quadrotor: meaning choosing settling time, overshoots, etc. A pole placement, is a possible solution. As for the rest, the deep neural network is in charge of learning the correction to be brought to the established linear controller so that it respects the specified requirements.

The obtained results confirm the methodology, both in simulation and in real-world experimentation. The proposed DNN-based controller allows to correct tracking. It overcomes model errors whether there are internal or external disturbances, such as ground effect or wind disturbance. The addition of the corrective term from the DNN allows to efficiently learn these effects. Hence, the DNN-enhanced controller fits the expected linear dynamic from the initial linear controller on the linear model.

However, the presented method has some disadvantages, some of which have already been stated:

- The presented approach is based on an offline learning of quadrotor dynamics. Thus, it is first needed to get a collection of relevant flights, before learning and then correcting the quadrotor behavior.
- The deep neural network is here dependent on the initial linear cascaded controller. Indeed, it is learning a dynamical error between obtained behavior and expected one with this initial controller. It is necessary to be able to flight with that initial controller on the given conditions.
- The closed-loop stability guarantees are limited. Indeed, it has been shown that proposed approach is working for test scenarios that are close to the training set. However, nothing ensures the stability for an unexpected case scenario completely different from the database. For instance, a trajectory generation that demands higher tracking velocity.
- It relies on accurate state estimation and acceleration of the quadrotor. Work must be carried out to better handle a bad estimation of these variables.

These disadvantages are opportunities to improve the proposed approach. In the next chapter, we handle some of these issues. In particular, including the closed-loop stability guarantee while learning online a deep neural network to correct the tracking.

Chapter IV

Towards an online implementation: event-based neural learning

This chapter is about improving the previously studied approach and adapting it to an online implementation requirement. After a brief introduction and a reminder of the models used, the event-based learning strategy is presented. Simulations are first performed to demonstrate the effectiveness of the criteria that compose it: stability and performance. Finally, flight experiments are performed to validate the strategy.

Contents

IV.1 Introduction

IV.1.1 Motivations

As already mentioned, data-driven techniques have already largely demonstrated their effectiveness. However, multiple problems are still raised concerning their applications in robotics.

First, such solution are often resource intensive algorithms. For instance RL [Str+06] is needing efficient data collection and large computation time. Indeed, such approaches when dealing with uncertain systems enforce the curse of dimensionality [BM03]. The number of parameters to learn become exponential with the size of the state. Solutions such as presented in [DO16], attempt to solve the problem. Concerning the efficient data collection solutions are also emerging as [Gle+22].

The second problem is the stability of the closed-loop system with such learning techniques. To overcome this issue, different techniques are currently studied. Some methods like [WZ21], rely on the use of safety filters. Thus, the control, whether it comes from a traditional approach or from a RL algorithm, is filtered by a module which guarantees that the system remains in a safe zone and so that the proposed controller does not destabilize or enter a zone considered as dangerous. Others integrate a safety layer to the learning process. In [Kol+19], proposed method guarantees the existence of trajectories leading to a safe region by estimating a statistical model of the uncertainties. The safety framework proposed by [Fis+17] is a mixed solution between model-based control and data-driven methods. It uses the knowledge of the dynamics to guarantee the satisfaction of the constraints. However, it still suffers from the curse of dimensionality.

Some solutions are more control oriented. For instance, the use Lyapunov function [Cho+18] or of Control Lyapunov Function to deal with uncertainties [Tay+19]. In [Tay+19] both parametric errors and unmodeled dynamics from Lyapunov function are learned via an episodic learning. It results in an improvement of the controller. Other approaches focus on Control Barrier Function [Cho+20]; [Wan+20], to handle the safe learning. They rely on learning dynamics or uncertainties presented in barrier functions. In [GT12], the authors uses reachability analysis to provide a safe learning framework. It is experimentally validated in a target tracking using a quadrotor.

Thus, a lot of solutions are emerging and proposing frameworks to ensure a safe learning. Concerning our approach, we decided to use the Lyapunov stability framework to guarantee stability of the closed-loop system, including the neural-based correction.

IV.1.2 Proposed event-based solution

To attempt to overcome aforementioned issues, we propose a mixed solution between standard control algorithm and learning approach, in an event-based manner. The proposed solution, named event-based neural learning strategy, is an in-flight learning algorithm based on the standard cascaded architecture. It is built upon a cascaded PD controllers easy to tune on any quadrotor. Once the traditional tuning step done, the proposed controller will automatically learn the corrections to be made to overcome the effects of encountered disturbances during the flight. This approach is avoiding the implementation of disturbance observers that needs a model of the disturbance and needs to be well tuned to achieve good tracking performance. The event-based learning strategy relies on a succession of data collection, learning and corrections phases. The correction is done by a deep neural network. During the learning time, the previously applied controller is still used to continue the mission without landing. Two criteria are implemented to ensure stability and tracking performance. The first one, the stability criterion, is based on Lyapunov and ensures that the DNN is not destabilizing the closed-loop system. The second one, the performance criterion, proposes updates to the DNN to continuously improve the tracking.

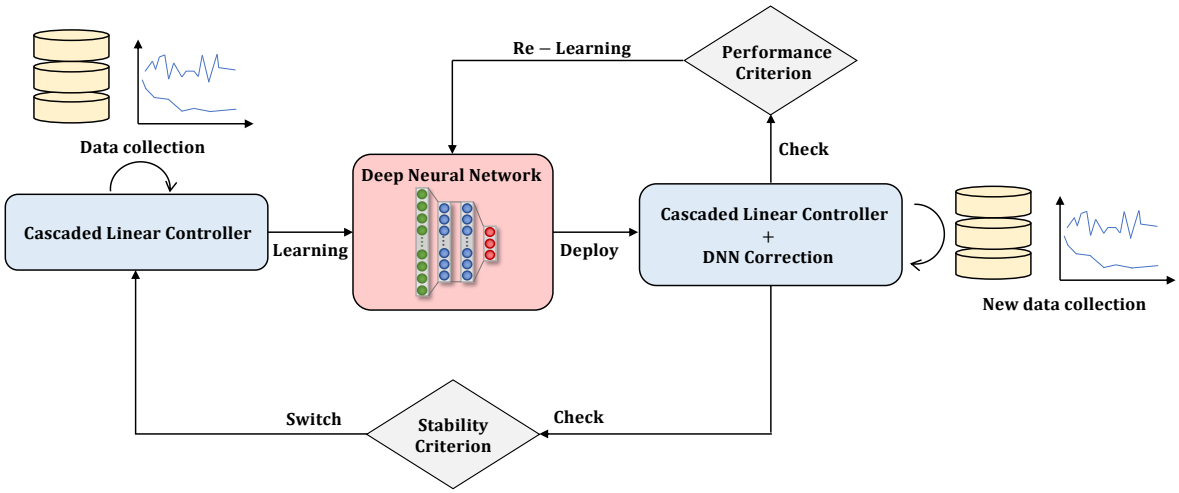


Figure IV.1: The proposed event-based neural learning strategy. At starting, an initial cascaded linear controller is used. It is enhanced with DNN learning after data collection. Two criteria are used to ensure both stability and flight tracking performance. If the stability is faulty, the control is switched to the initial controller.

The proposed event-based neural learning methodology is summarized in Fig. IV.1. It is composed of the following elements:

- **Initial setup:** Before flying, the initial cascaded controller is designed according to specified linear expected behavior.
- **Data collection and iterative learning:** After a given flight time duration and a specified number of data to be collected, a first DNN is trained. It learns errors

made during the flight. It is integrated in the control scheme to correct the flight behavior at the end of its learning. During, the learning time the previous controller is still running and collecting data. The DNN is iteratively trained according to the performance criterion.

- **Performance criterion:** If the trajectory tracking using the given DNN is not satisfactory in term of trajectory tracking, the DNN is improved by re-learning. The strategy will refine the DNN gains using the newly collected data. The relearning is triggered when the criterion on all three spatial positions is exceeded.
- **Stability criterion:** If the proposed DNN is not safe according to an established Lyapunov-based criterion, the strategies switches to the initial controller. It then collects and learns another DNN using newly collected data.

In the following, we first recall the initial cascaded controller and the different changes made in comparison to the previous chapter in section IV.2. The event-based neural learning strategy is explained in section IV.3. Then, it is tested and validated in simulation and experimentally in section IV.4.

IV.2 Recall and changes introduced

This first section has two main purposes. Firstly, we recall the initial position and velocity controller used for the proposed strategy. It lies on the previously established controller but it includes some adjustments. It is presented in the subsection IV.2.1. Secondly, we explain the changes made to the DNN architecture. We also present modifications brought to the DNN learning. It is done in the subsection IV.2.2.

The changes concerning the event-based neural learning strategy using these elements, initial controller and DNN, will be done in the following section IV.3.

IV.2.1 Initial cascaded controller

As for the solution presented in previous chapter III, the initial controller is based on the standard cascaded architecture for position, velocity and attitude. That controller does not include yet a neural feedforward term. Again for simplicity of tuning, we decided to keep the PD controller for the position and velocity control.

The proposed initial position and velocity controller has the same structure as (III.7). We recall here the expression of the control command:

$$u_\zeta^* = -\boldsymbol{K_\eta}\eta + \boldsymbol{B_\zeta^{-1}}\dot{\tilde{v}}_{ref}, \tag{IV.1}$$

where:

- η is the state error variable composed of position and speed errors,
- u_ζ^* is the command input composed of desired thrust, desired pitch and roll angles. That command is send to the inner attitude controller, it is kept unchanged (see III.4).
- $\boldsymbol{B_\zeta^{-1}}\dot{\tilde{v}}_{ref}$ is a term to compensate acceleration from the trajectory generation,
- $\boldsymbol{K_\eta}$ is a matrix gain to achieve desired behavior on position and velocity tracking.

The gain $\boldsymbol{K_\eta}$ was previously designed using a linear quadratic regulator synthesis. This is no longer the case here, it is given by the solution of a Ricatti equation, to match expected requirement on stability. It is latter explained in section IV.3.2 and given by (IV.12).

IV.2.2 Error dynamics learning modifications

This section presents the proposed structure for the neural estimation of δ_t. Is is based on previously explained architecture III.3. Some modifications were brought:

The architecture of the neural network remains unchanged compared to (III.10) but the number of units that compose its hidden layers has been reduced. Here, the number of intermediate neurons has been reduced to 32. The motivation for this change is threefold. Firstly, the dynamics of the quadrotor alone can be correctly learned with 32 units per layer using the approach presented in the previous chapter. The second motivation comes from the fact that the databases on which the error dynamics will be learned are smaller. Thirdly, it will allow a faster learning for the online application.

Modification of the input of the neural network: We modified the input of the neural network compared to previously established network (III.11). It is now given by:

$$\mathrm{x}_t = \begin{bmatrix} c_\phi, & s_\phi, & c_\theta, & s_\theta, & v_x^2, & v_y^2, & v_z^2, & u_{batt}, & T \end{bmatrix}^T \tag{IV.2}$$

Two modifications were brought: first we provided squared velocities indeed it appears in the drag force (I.12). It will be easier to learn it providing directly squared velocities instead of simple one. Next, we removed z component which was introduced to help the learning of ground effect. It is mainly motivated as we don not focus on learning ground effect, and maintaining it makes the learning more complex: slower convergence for same hyper-parameters.

The learning is now done online during flights. It is done over several small databases collected and created during the quadrotor's flights. The collection and learning procedures are ruled by the proposed event-based neural learning strategy explained in section IV.3. During the learning, the current DNN-based controller is used. At the end of the training, it switches to the new one. Each training set includes thirty seconds of collected

data $(\mathbf{x_k}, \mathbf{y_k})$. In the following sections, $\hat{\delta}_t^k$ will denote the k-th DNN learned. It means the k-th update of the network using the newly collected database. Thus, the $(k+1)$-th network has the same structure as the k-th one, but with newly learned weights and biases. These new parameters are estimated from k-th weights and biases and are refined using the new database.

The loss function and the optimizer remain unchanged. The MSE is still used as loss function. Again, NADAM algorithm is used but with some parameters adjustments. The batch size, number of epochs and learning rate are empirically modified to achieve a fast and good learning of δ_t in the considered situations. The list of parameters and hyper-parameters are summarized in Table IV.1.

Parameter	Value
Number of inputs #x	9
Number of hidden layers n_l	2
Number of units n_u	32
Collection time (s)	30
Batch	128
Epoch	250
Learning rate l_r	0.005
μ	0.9
ν	0.9999

Table IV.1: List of parameters and hyper-parameters selected for the DNN and its training.

Having outlined the initial controller and the modifications made to the learning, the event-based neural learning strategy can now be presented.

IV.3 The event-based neural learning strategy

This section aims to present the event-based neural learning strategy. The objectif of this strategy is to continuously learn and adapt the control in order to track a reference trajectory while facing internal and external disturbances. It is based on the previously presented cascaded architecture to achieve position & velocity control. First, the DNN-based correction term, which is used to improve the initial controller, is presented in section IV.3.1. Next, the event-based neural learning strategy, which is designed to ensure stability and improve tracking performance, is then discussed in section IV.3.2. This strategy initiates a new learning process when tracking errors exceed a certain threshold: it is the first event, defining the criterion named **performance criterion**. It switches the controller if the DNN destabilize the quadrotor. The second event is based on the crossing of a criterion on a Lyapunov function, it is referred as **stability criterion**.

IV.3.1 DNN-based controller

The real system dynamics, including the notation from chapter III, are revisited here:

$$\dot{\eta} = \overline{\boldsymbol{A}}\eta + \overline{\boldsymbol{B}}(u_\zeta - \boldsymbol{B_\zeta^{-1}}\dot{\tilde{v}}_{ref} + \boldsymbol{B_\zeta^{-1}}\delta_t). \tag{IV.3}$$

This model has a linear and a nonlinear part. The nonlinearity term includes internal and external disturbance via δ_t dynamics. In order to cancel the nonlinearities, the proposed DNN-based controller used is given by:

$$u_\zeta = u_\zeta^* + \Delta u_\zeta^k, \tag{IV.4}$$

where u_ζ^* is the linear control term (IV.1) and Δu_ζ^k is the feed-forward correction term based on the k-th learned deep neural network.

The expression of the closed loop dynamics, using proposed controller can now be developed using (IV.3) and (IV.4):

$$\dot{\eta} \;=\; \underbrace{(\overline{\boldsymbol{A}} - \overline{\boldsymbol{B}}\boldsymbol{K_\eta})\eta}_{\text{Closed-loop linear dynamics}} \;+\; \underbrace{\overline{\boldsymbol{B}}(\Delta u_\zeta^k + \boldsymbol{B_\zeta^{-1}}\delta_t)}_{\text{Term to cancel}}. \tag{IV.5}$$

The closed-loop dynamics is the sum of two terms:

1. The closed-loop linear behavior also referred as desired behavior. It is determined by $\boldsymbol{K_\eta}$ designed to achieved performance specifications using a Ricatti equation.
2. An additional nonlinear term that can be handle via Δu_ζ^k. As the whole term vanishes, the closed-loop dynamics approaches the desired linear behavior.

To cancel the second term, an optimization problem must be solved, it is given by the following:

$$\min_{\Delta u_\zeta^k} \left\| \Delta u_\zeta^k + \boldsymbol{B_\zeta^{-1}}\delta_t \right\|^2 \tag{IV.6}$$

Since the non-linearity term is unknown in (IV.6), it can be replaced by its current estimate from the DNN $\hat{\delta}_t^k$:

$$\min_{\Delta u_\zeta^k} \left\| \Delta u_\zeta^k + \boldsymbol{B_\zeta^{-1}}\hat{\delta}_t^k(\mathrm{x}_t) \right\|^2 \tag{IV.7}$$

The minimization problem given in equation (IV.7) is a nonlinear optimization since $\mathrm{x}_t = f(u_\zeta) = f(u_\zeta^* + \Delta u_\zeta^k)$. Indeed, the input x_t depends on the DNN correction term Δu_ζ^k which is a function of the error estimate $\hat{\delta}_t^k$. Finding the solution to this nonlinear and non-convex minimization problem can be time-consuming and may have multiple local minima.

The proposed solution involves providing the neural estimate with only the linear control input, as in previous chapter. It corresponds to replace u_ζ by u_ζ^* in the DNN input in

eq. (IV.7). By removing the correction term from the DNN input, the solution can be directly obtained:

$$\Delta u_\zeta^k = -\boldsymbol{B_\zeta^{-1}} \hat{\delta}_t^k(u_\zeta^*). \tag{IV.8}$$

Thus, as $\boldsymbol{B_\zeta^{-1}}$ matrix is fully known and as the quadrotor total mass is known, the correction to be applied to the thrust T, ϕ_{ref} and θ_{ref} angles references can be quickly computed. The complete cascaded architecture proposed is summarized in Fig. IV.2.

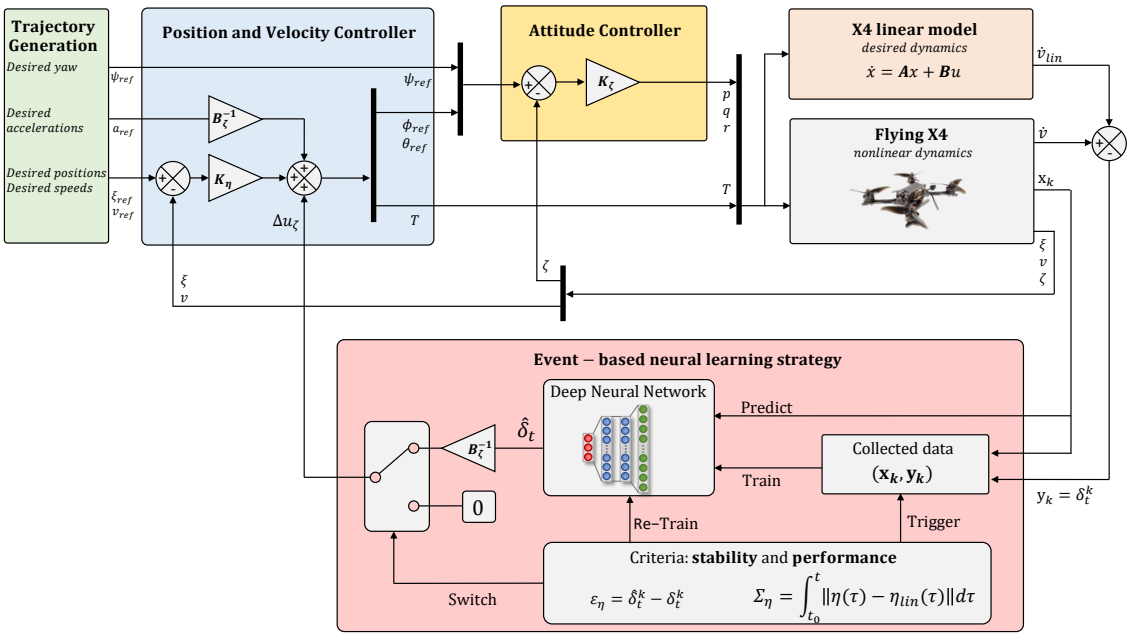


Figure IV.2: The event-based neural learning strategy: cascaded control architecture.

One needs to highlight the fact that the correction is only computed for position and velocity loop. The attitude loop corresponds to the frame rotations and does not involve any disturbances that need to be estimated.

With the control input now fully defined, the event-based neural learning strategy can be introduced. It decides to learn or re-learn $\hat{\delta}_t^k$, according to stability and to performance criteria.

IV.3.2 The event-based strategy

As previously noted, the DNN-based control term given in (IV.4) depends on the estimate of $\hat{\delta}_t^k$. Therefore, the accuracy of this estimation is crucial for maintaining the stability and performance of trajectory tracking. This estimate needs to be updated regularly. To achieve this, updates are performed using small dataset collected during the flight. Since the quantity of data is limited and may not contain sufficient information for an accurate global approximation, this term must be updated frequently using different criteria.

If the DNN learning is not accurate, the quadrotor may not perform as expected. Two main events can be expected, defining two criteria:

1. **Stability criterion**: if the estimate $\hat{\delta}_t^k$ is not sufficiently accurate, it may compromise the stability of the closed-loop system. It is necessary to first detect this critical situation using a stability criterion. The proposed solution in this case is to immediately switch to the initial linear controller, without any correction term by applying $\Delta u_\zeta^k = \mathbb{0}_3^T$. Afterwards, a new data collection procedure is performed in order to estimate a more accurate DNN. The design of the stability criterion is explained in the sub-section IV.3.2.1.

2. **Performance criterion**: even if the predictions of the DNN are not as accurate as desired, it may not cause any stability issues and not be detected using previous criterion. For instance, a static error in tracking. To deal with these situations, the proposed solution is to re-learn $\hat{\delta}_t^k$ getting a new and more accurate estimate: $\hat{\delta}_t^{k+1}$. After the training and update of the network, the DNN-based correction term applied is given by $\Delta u_\zeta^{k+1} = -\boldsymbol{B}_\zeta^{-\mathbf{1}} \hat{\delta}_t^{k+1}(u_\zeta^*)$. The performance criterion is explained in the sub-section IV.3.2.2.

The global scheme of the event-based neural learning approach is summarized in Fig. IV.2 and the overall methodology in Fig. IV.1. In the following, detailed explanations of both stability and performance criterion are given.

IV.3.2.1 Stability criterion

In order to establish a switching criterion based on stability, it is needed to evaluate the overall closed-loop stability of the quadrotor. We use Lyapunov's stability definition to assess the stability of the system.

Let $\mathcal{V}$ be the candidate Lyapunov function:

$$\mathcal{V} := \eta^T \boldsymbol{P} \eta, \tag{IV.9}$$

where $\boldsymbol{P}$ is the solution of the following Riccati equation:

$$\bar{\boldsymbol{A}}^T \boldsymbol{P} + \boldsymbol{P}\bar{\boldsymbol{A}} - 2\boldsymbol{P}\bar{\boldsymbol{B}}\bar{\boldsymbol{B}}^T \boldsymbol{P} = -\alpha \boldsymbol{P}. \tag{IV.10}$$

The equation (IV.10) can be equivalently written as:

$$\left(\bar{\boldsymbol{A}} + \frac{\alpha}{2}\boldsymbol{I}\right)^T \boldsymbol{P} + \boldsymbol{P}\left(\bar{\boldsymbol{A}} + \frac{\alpha}{2}\boldsymbol{I}\right) - 2\boldsymbol{P}\bar{\boldsymbol{B}}\bar{\boldsymbol{B}}^T \boldsymbol{P} = 0. \tag{IV.11}$$

Let $\boldsymbol{K_\eta}$ be given by:

$$\boldsymbol{K_\eta} := \bar{\boldsymbol{B}}^T \boldsymbol{P}, \tag{IV.12}$$

Replacing (IV.1) in (IV.3), it gives:

$$\dot{\eta} = \overline{\boldsymbol{A}}\eta + \overline{\boldsymbol{B}}(-\boldsymbol{K_\eta}\eta + \Delta u_\zeta^k + \boldsymbol{B_\zeta^{-1}}\delta_t). \tag{IV.13}$$

Using (IV.8) and defining $\varepsilon_\eta := \delta_t - \hat{\delta}_t^k$, from (IV.13) it follows that:

$$\dot{\eta} = (\overline{\boldsymbol{A}} - \overline{\boldsymbol{B}}\boldsymbol{K_\eta})\eta + \overline{\boldsymbol{B}}\boldsymbol{B_\zeta^{-1}}\varepsilon_\eta. \tag{IV.14}$$

The time derivative of the Lyapunov function is then given by:

$$\dot{\mathcal{V}}(t) = \dot{\eta}^T\boldsymbol{P}\eta + \eta^T\boldsymbol{P}\dot{\eta}. \tag{IV.15}$$

Replacing the dynamics (IV.14) in (IV.15), one obtains the following expression:

$$\dot{\mathcal{V}}(t) = \eta^T\left((\bar{\boldsymbol{A}} - \bar{\boldsymbol{B}}\boldsymbol{K_\eta})^T\boldsymbol{P} + \boldsymbol{P}(\bar{\boldsymbol{A}} - \bar{\boldsymbol{B}}\boldsymbol{K_\eta})\right)\eta + 2\eta^T\boldsymbol{P}\bar{\boldsymbol{B}}\boldsymbol{B_\zeta^{-1}}\varepsilon_\eta. \tag{IV.16}$$

From (IV.16), we finally obtain the expression:

$$\dot{\mathcal{V}}(t) = -\alpha\mathcal{V} + 2\eta^T\boldsymbol{P}\bar{\boldsymbol{B}}\boldsymbol{B_\zeta^{-1}}\varepsilon_\eta. \tag{IV.17}$$

Let $\boldsymbol{P^{\frac{1}{2}}}$ denote the square matrix such that $\boldsymbol{P^{\frac{1}{2}}}\boldsymbol{P^{\frac{1}{2}}} = \boldsymbol{P}$. It should be noted that because the system is controllable, $\boldsymbol{P}$ is symmetric positive definite. Thus, $\boldsymbol{P^{\frac{1}{2}}}$ always exists and can be chosen to be also symmetric positive definite. We thus introduce $w := \boldsymbol{P^{\frac{1}{2}}}\eta$, then eq. (IV.17) becomes:

$$\dot{\mathcal{V}} = -\alpha w^T w + 2w^T\boldsymbol{P^{\frac{1}{2}}}\bar{\boldsymbol{B}}\boldsymbol{B_\zeta^{-1}}\varepsilon_\eta. \tag{IV.18}$$

Using the norm operator on (IV.18), we have:

$$\dot{\mathcal{V}} \leq -\alpha\left\|w\right\|^2 + 2\lambda_{\max}(\boldsymbol{P^{\frac{1}{2}}}\bar{\boldsymbol{B}}\boldsymbol{B_\zeta^{-1}})\left\|w\right\|\left\|\varepsilon_\eta\right\|. \tag{IV.19}$$

Therefore, as long as the norm of ε_η is sufficiently small, the Lyapunov function will be decreasing. More precisely, since $\|w\| = \sqrt{\mathcal{V}}$, one can write (IV.19):

$$\|\varepsilon_\eta\| < \frac{\alpha}{2\lambda_{\max}(\boldsymbol{P^{\frac{1}{2}}}\bar{\boldsymbol{B}}\boldsymbol{B_\zeta^{-1}})}\sqrt{\mathcal{V}}. \tag{IV.20}$$

If (IV.20) is satisfied, then $\mathcal{V}$ will be strictly decreasing. A threshold variable (IV.21) can be defined:

$$\varepsilon_m = \frac{\alpha}{2\lambda_{\max}(\boldsymbol{P^{\frac{1}{2}}}\bar{\boldsymbol{B}}\boldsymbol{B_\zeta^{-1}})}\sqrt{\mathcal{V}}. \tag{IV.21}$$

Then if, we get this matching condition:

$$\|\varepsilon_\eta\| - \varepsilon_m \geq 0, \tag{IV.22}$$

the controller is switched to the initial one, given in eq. (IV.1) since the defined Lyapunov function (IV.9) is not guaranteed to decrease anymore. In practice, switching the controller is done by applying $\Delta u_\zeta^k = \mathbb{0}_3^T$. Afterwards, a new data collection is performed using this

linear controller. When a sufficient amount of data has been collected, a new DNN is trained and then applied using equation (IV.8). This event is based on the evolution of $\|\varepsilon_\eta\|$, it is computed thanks to the output of the DNN and the acceleration value obtained from the IMU. The process is then repeated until the stability criterion is triggered again, indicating that the newly learned DNN is unable to guarantee stability in the sense of Lyapunov.

IV.3.2.2 Performance criterion

It is possible that errors resulting from an inaccurate estimation of $\hat{\delta}_t^k$ may not compromise the stability of the quadrotor. Therefore, a second criterion must be defined to address this type of scenario. Indeed, the first criterion given in (IV.22) may not be triggered while the tracking performance is not satisfactory. For instance, in the presence of a static error not corrected by the neural network: the output of the DNN is constant and is not triggering the stability threshold. Then, the quadrotor will continue to displace with poor a tracking performance. To overcome this problem, it is proposed to use a performance criterion based on the cumulative error to the linear model:

$$\Sigma_\eta(t) = \int_{t_s}^{t} \|\eta(\tau) - \eta_{lin}(\tau)\|^2 \, d\tau \tag{IV.23}$$

Then a new learning is triggered when:

$$\Sigma_\eta(t) > \Sigma_m. \tag{IV.24}$$

where Σ_m is a threshold experimentally chosen so as to trigger a new learning when there is enough error to be corrected. This criterion is coupled with the fact that sufficient data must also be collected. To prevent *catastrophic forgetting* [Rob95], meaning the complete forgetting of previously learned tasks, a small part of the previously built database is kept and injected in the new one. That procedure is called *rehearsal*. Moreover, the learning rate can be decreased using a multiplying factor at each new re-learning trigger.

In the next section, we illustrate the event-based neural learning strategy. It is tested in simulation environment and then on real flights tests.

IV.4 Simulation and experimental results

In this section, the proposed strategy is first validated in the Gazebo simulation environment in section IV.4.1. The two previously defined criteria are tested and analyzed. Next, the event-based neural learning strategy is tested in real-world in section IV.4.2. The details about the experimental setup is still unchanged and given in the dedicated chapter II.

IV.4.1 Simulations results

The simulations are performed using Gazebo environment. In order to validate the proposed strategy, two independent scenarios are proposed:

- (I) - A first scenario: it involves repeating a series of circles in x/y plane,
- (II) - A second scenario: it requests the quadrotor to first perform several circles in x/y plane and next to perform steps in y/z plane.

For both tests, no external disturbances are included to the simulation environment. Thus, in these test case scenarios, errors are mainly cause because of the initial linear cascaded controller (IV.1). Indeed, there is no integral action in this controller, to keep simplicity of tuning but also because in the perfect linear case it should be sufficient enough. Throughout the different scenarios, the quadrotor will learn to adjust its behavior adopting the proposed strategy.

IV.4.1.1 Performance criterion analysis: scenario (I)

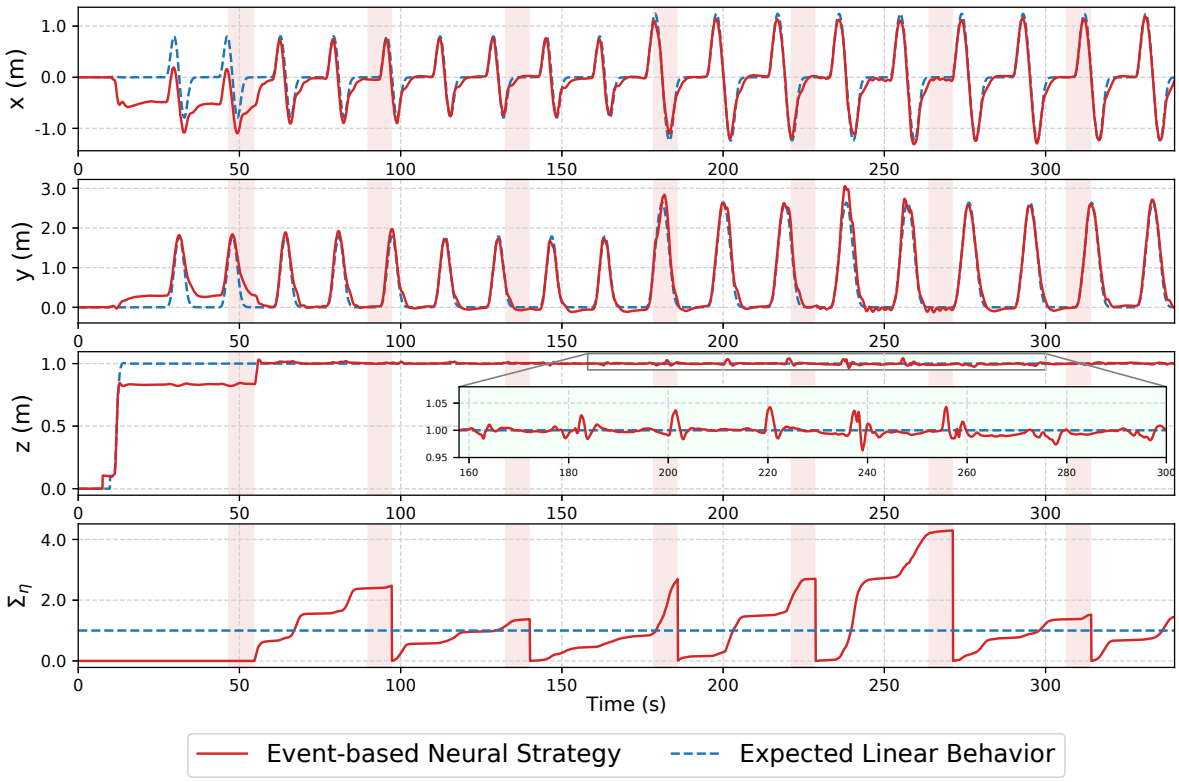


Figure IV.3: Simulation scenario (I): circles motions. This scenario illustrates the performance criterion. Red areas represent learning and re-learning processes.

A first scenario, that consists in successive circle motions is firstly performed to assess the performance criterion. The simulations results of this scenario (I) are given in Fig. IV.3. The first three plots represent the spatial motions along x, y and z. The fourth graph is representing the cumulative error representing the tracking performance (IV.23). Learning procedures are represented in pink bands. The performance criterion is reset after each re-learning.

At starting, major static errors can be noted in all three spatial positions. It is justified by the lack of integrator in the position controller. A first data collection, together with a first DNN, is proposed after few seconds of flight. It corrects the tracking and brings the quadrotor closer to the linear desired behavior, represented in blue dashed dot. The tracking error is smaller, but it can still be improved since the deep neural network has not yet fully learned the dynamics to be corrected. The cumulative error, (IV.23), is directly increasing after deploying the new DNN. After a second data collection and when the performance threshold has been exceeded, a re-learning procedure is initiated. A second DNN, improved version of the first proposed DNN, is then applied around $t = 90s$. The tracking performance is improved and the criterion is increasing slower. At $t = 180s$, the trajectory generation requires the quadrotor to perform larger circles. These motions were not part of the previous training scenario, resulting in a less accurate tracking. However, it is worth noting that main static errors of the initial controller are corrected. The learning procedure is triggered once more and refines the DNN proposed and so forth. At the end, the quadrotor behavior is significantly closer to the expected linear behavior.

IV.4.1.2 Stability criterion analysis: scenario (II)

A second test scenario (II) is proposed to validate the stability criterion. Results are given in Fig. IV.4. Here, the procedure is restarted from scratch, where only the initial linear cascaded controller is used at starting and no DNN is proposed yet. For experimental reasons, the criterion given in eq. (IV.22) is implemented with a time-triggering threshold. Indeed, in order to avoid possible unwanted and unnecessary switching due to noise, the controller switch is activated if the stability criterion is activated during ten time steps. The first three plots illustrate the displacement on x, y and z. The second one is representing the cumulative error (IV.23) and the last curve represents the stability criterion (IV.22).

This new test scenario is composed of eight circles motions in x/y plane followed by six steps in y/z plane after $t = 160s$ of flight. As for the previous scenario, the quadrotor starts with large static errors that are next corrected using successive improvements of $\hat{\delta}_t^1$ obtained by the previously studied performance criterion. The DNN is re-learned up to $\hat{\delta}_t^4$. After $t = 160s$, the quadrotor is asked to perform a succession of steps. Such test cases were not part of the training dataset of the neural controller. It starts to well behave, however, at $t = 180s$, the stability criteria (IV.22) is triggered. It means that the lastly proposed DNN, $\hat{\delta}_t^4$, is destabilizing the closed-loop system according to our criterion. The initial linear cascaded controller is immediately used to maintain the stability and collect new pertinent data to learn a novel DNN. After that training, the proposed DNN, $\hat{\delta'}_t^0$, is correcting the

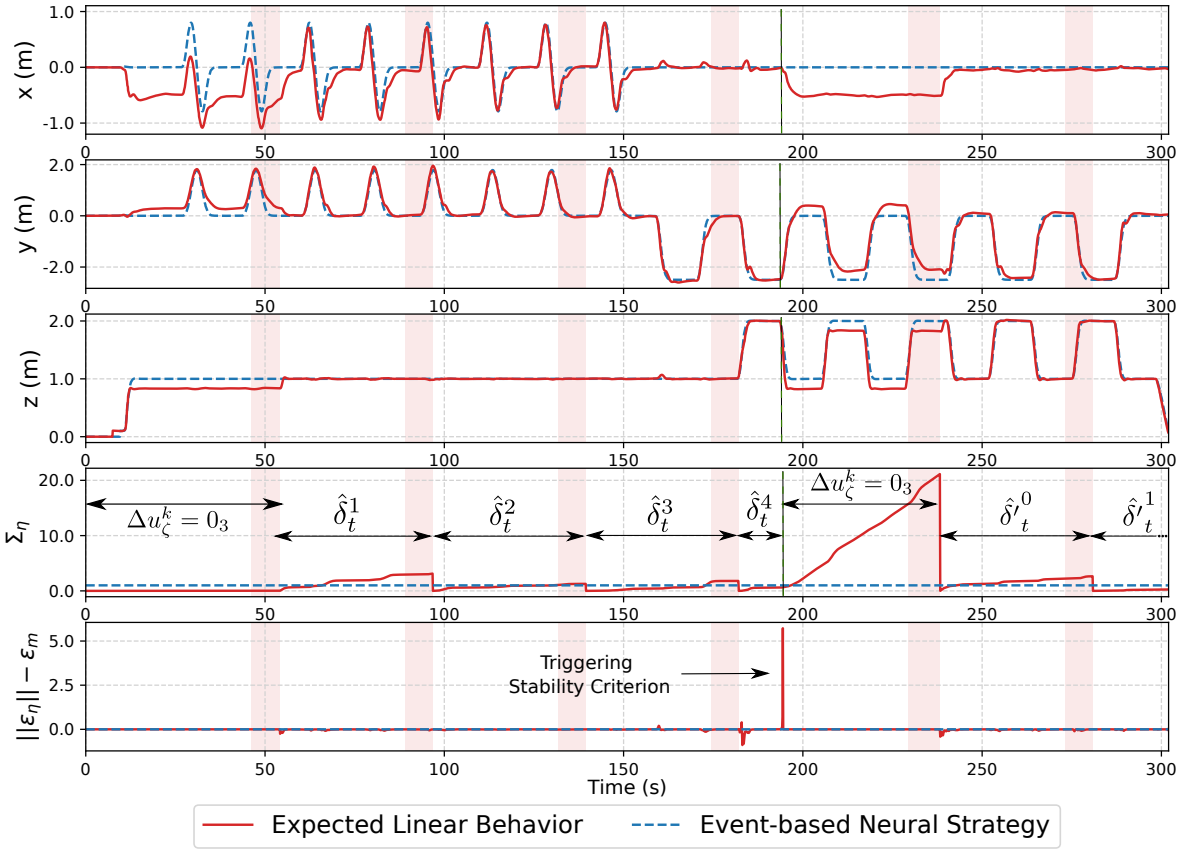


Figure IV.4: Simulation scenario (II): circles followed by steps motions. This scenario illustrates stability and performance criteria. Red areas represent learning and re-learning processes.

tracking to fit the linear behavior correctly without destabilization during the step motions.

The choice to switch to the initial controller instead of the previously learned DNN was motivated by the fact that when testing that other solution, the proposed strategy was successively switching the DNN controllers, $\hat{\delta}_t^N$ to $\hat{\delta}_t^0$, till the initial controller. Indeed, if the lastly learned deep neural network is not able to well behave, there is a high possibility that the previous DNN has not learn this information.

Now, as the proposed learning strategy and its two criteria were validated in simulation, one can deploy it in real tests, in the next section.

IV.4.2 Experimental results

In order to experimentally validate the contribution of the event-based neural learning strategy, we performed experimentation under external disturbances. Here, experimental tests are performed in a windy environment generated by a wind indoor tunnel. The quadrotor is asked to face a sinusoidal wind doing back and forth motions, as illustrated in Fig. IV.5. We

recall that other experimental aspects are given in chapter II.

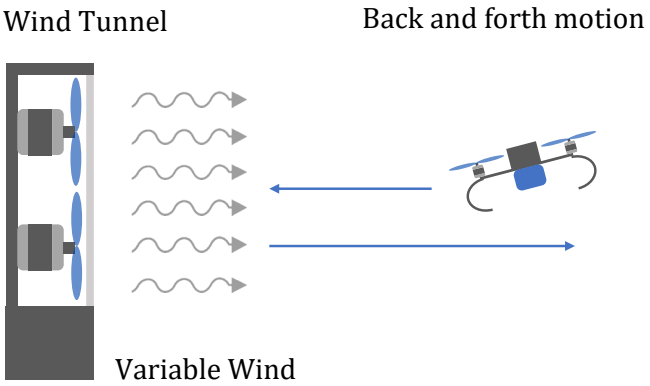


Figure IV.5: Experimental test illustration: back and forth motions in front of a sinusoidal varying wind.

Following results are compared to a well tuned PID controller coupled with a disturbance observer. That controller was previously tuned in order to fit the desired expected linear closed-loop behavior given by the PD controller without DNN correction (IV.4) in the windy condition. The experimental results are provided in Fig. IV.6. The four first plots respectively represent, x, y and z positions, performance criterion and stability criterion. The last curve is representing the intensity of the wind applied, it is given in meter per second. It is an approximate value calculated from the maximum speed of the wind generated by the wind indoor tunnel and the transmitted value in PWM. Applied disturbance is a sinus whose frequency is around 0.046Hz starting at $t = 10s$. It is important to notice that the wind frequency is not synchronized with the back and for motion of the quadrotor.

When the wind begins, in part n°0 in the figure, the proposed controller is not able to correctly reject the disturbance. it hence large errors compared to the PID cascaded controller along with the disturbance observer. After a collection time, and a learning, the first DNN mainly overcomes the wind disturbance effect, in part n°1. In part n°2, the quadrotor has re-learned the deep neural network, but it still presents difficulty in z component compared to the other controller. Indeed, the battery discharge is not yet well learned.

A change of amplitude and offset intensity of the wind is operate at $t = 145s$, at the end of part n°2. It can be noted that the proposed controller with the previously learned controller has difficulty to reach its desired target point. Indeed the intensity of the wind is stronger and was not previously learned. After a new collection and re-learn process, the quadrotor is able to get closer to the linear desired behavior in part n°4 and n°5.

It is worth noting that when applying the same DNN, the behavior between each steps is not always the same. Indeed, conditions are varying at each step: the de-synchronized wind, the increased battery discharge, the wind disturbance generated in the whole room, etc. Thus, the output of the network is not exactly the same each step iteration and the correction proposed by the controller is consequently slightly different, leading to a different behavior at each new step. The Root Mean Squared Error (RMSE) of all parts [0 to 5]

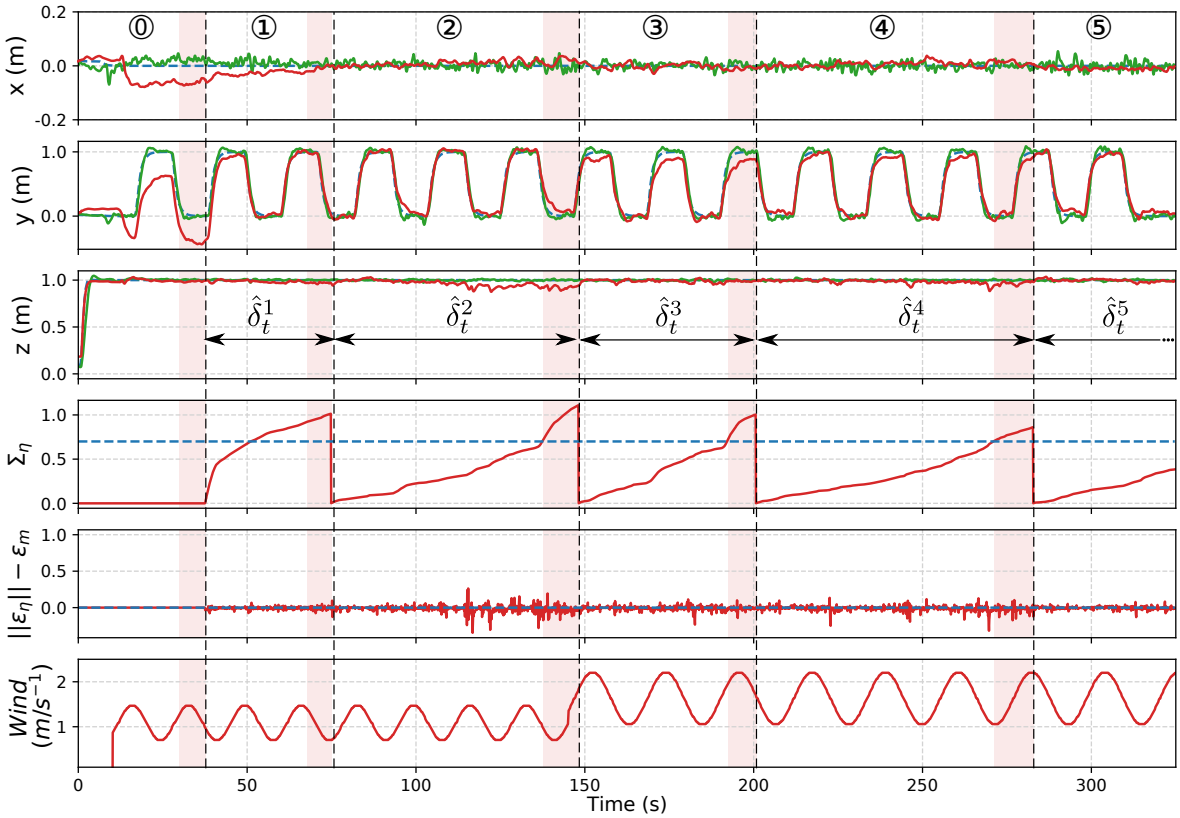


Figure IV.6: The event-based neural learning strategy experimental test: back and forth motions in front of a sinusoidal varying wind. Red areas represent learning and re-learning process. For x, y and z, the red curve is the result with the event-based neural learning strategy. The green curve is the obtained result with the PID along with disturbance observer. The blue one is the expected linear behavior with the linear PD controller only.

Figure IV.7: The Kopis quadrotor facing the wind indoor tunnel.

presented in the figure are computed for both controllers compared to the linear expected behavior with PD controller in a perfect non windy case. RMSEs are provided in Table IV.2. The RMSE general formula is recalled in eq. eqIV.25.

$$RMSE = \sqrt{\sum_{i=1}^{N_s} \frac{(x_i - y_i)^2}{N_s}} \tag{IV.25}$$

where N_s is the number of sample considered, x_i the expected value and y_i the obtained one.

	Event-based Neural Learning			PID+Observer		
Part	x	y	z	x	y	z
0	0.054	0.323	0.02	0.023	0.046	0.072
1	0.031	0.098	0.021	0.015	0.061	0.006
2	0.012	0.065	0.053	0.013	0.058	0.006
3	0.011	0.114	0.018	0.012	0.06	0.007
4	0.011	0.059	0.027	0.012	0.064	0.007
5	0.013	0.059	0.015	0.016	0.064	0.064

Table IV.2: RMSE (m) in each components during the wind scenario for each controller.

A global improvement of RMSE through flights, can be observed with the proposed event-based neural learning strategy. It assess a continuous improvement of the tracking. The behavior in z-component of the PID along with the disturbance observer is globally better than the proposed solution. Indeed, the proposed strategy does not include integral action and is progressively learning the battery discharge profile. It needs more time to better learn the correction. However, at the end of the scenario, it is improved as more data as been used and introduced in the lastly proposed DNN. Also, the PID controller along with the observer produces more low frequency oscillations during whole scenario in each component. Finally, it is worth noting that the stability criteria is very noisy, but via the filtered procedure, it is not triggered during the proposed scenario.

Finally, one presents in Fig. IV.8 an example of stability triggering in an experimental test. The quadrotor was asked to perform back and forth motion in front of the wind indoor tunnel, with a stronger initial wind. Few seconds after the first learning, the strategy switches to the initial controller because the threshold on stability has been crossed. A new data collection is performed and then the strategy proposes a new DNN. Next, the controller corrects the trajectory tracking using the performance criterion, as in the previous test.

A video of the experimental results is available via the following link: `youtu.be/S4I2dlPDCJY`

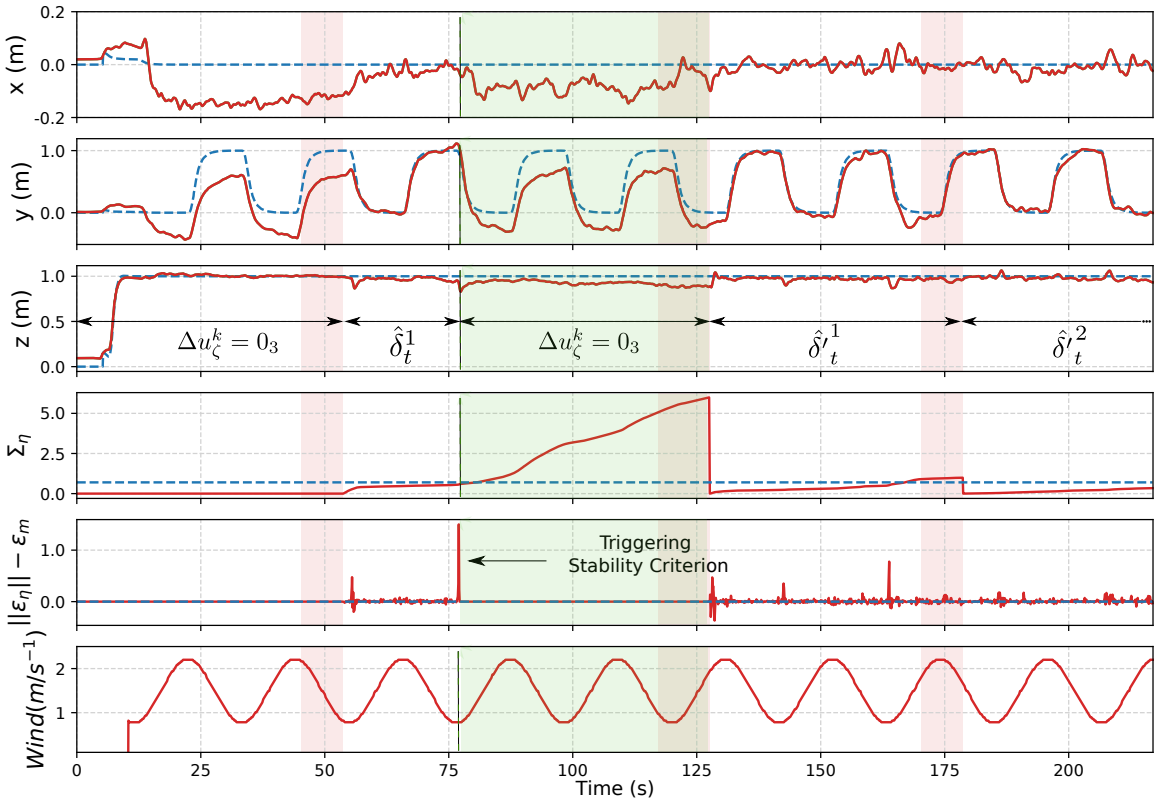


Figure IV.8: The event-based neural learning strategy experimental test: illustration of the stability criterion.

IV.5 Conclusion and perspectives

In this chapter, a solution to deal with an online DNN-based controller was proposed to overcome both internal and external disturbances. The design of that controller was subject to several constraints. First, to allow an easy implementation on standard controller while maintaining efficient trajectory tracking. Second, to keep the closed-loop system stable under the DNN-correction. Third, the solution has to be implemented online avoiding an offline learning.

The proposed controller is an event-based solution that is iteratively learning a DNN to correct the tracking. It is based on successive data collection and learning procedures. The event-based strategy allows the quadrotor to flight while learning or relearning a new controller. In that manner, it avoids landing before testing a new DNN. The strategy relies on two criteria. A performance criterion is employed to evaluate the tracking performance and triggers a DNN re-learning if the tracking performance is unsatisfying. A stability criterion is included in order to evaluate the closed-loop stability of the quadrotor. It switches to the initial controller in case of instability due to the imprecise learning of the DNN. The proposed strategy is working both in simulation and experiment. It demonstrates its effectiveness in case of external disturbances such as sinusoidal wind.

Some improvements can already be stated. Indeed, the initial controller does not include integral term for two reasons: first because in the case of a "perfect" linear behavior it is not needed. Also because of the tuning simplicity to reduce the number of gains to be tuned. A first step would be the adaption of the proposed strategy using an integral term. It might be need to give the integral action to the network input. In addition, both controllers must be adapted so that they can interact correctly without trying to correct each other.

The solution proposes to switch to the initial controller in case of lack of stability insurance. After that, a new learning is proposed. It needs to learn a new DNN. Thus, previously learned knowledge must be relearned, and it can represent a waste of time. Some work must be carried out in order to validate and definitely confirm a learning. A possible solution might be the use of several parallel networks used in different situations, a type of "DNN scheduling".

Chapter V

Neural controller for unknown payload transportation

This chapter focuses on the unknown suspended payload transportation problem. This task requires a more sophisticated controller to be correctly handled. The proposed controller is a DNN-assisted exact linearization After an introduction of the problematic, the proposed nonlinear controller is described. Then, the payload dynamics estimation using a DNN is exposed. Finally, experimental transportation is performed to validate the solution.

Contents

V.1 Introduction

In the two previous chapters, we focused on learning external dynamics such as wind disturbance or ground effect. Both presented control scheme still present issues. The first one is the linear initial controller that limits the flight possibilities. With higher velocities, the linear controller might become in adapted. Indeed, some of the neglected effects are no longer small enough, the quadrotor is moving from its equilibrium. The second one, is that the proposed solutions are not suitable for more concrete application cases such as load transportation scenarios. In such cases, internal disturbances generated by the payload can be very difficult to handle with previous presented controllers. Thus, a new controller must be developed.

V.1.1 Delivery application solutions

Quadrotor payload transportation is used for many delivery applications, from simple home delivery through first aid deployment to rescue at sea, by bringing a rescue buoy. These uses of the quadrotor are increasingly employed because it allows: a saving in delivery time, access to hard-to-reach areas or even reduction of emissions. Payload transportation can be achieved by several ways. First, the package can directly be attached to the frame of the device employing fastener. Another way, to perform it, is to use a robotic arm attache to the quadrotor for precise manipulation. Finally, it can be hooked to a suspended cable attached to the frame of the quadrotor, compared to the first one it reduces the disturbances locally generated by the proximity of the load from the motors. However, the additional suspended payload is acting as a pendulum attached to the device. This is a very challenging area of research, as modeling and control become even more complicated. Actually, the suspended payload is generating additional disturbances to the device. Notably during acceleration or deceleration, it can generate oscillations that can damage the package, or even worth, the whole system stability. Consequently, the influence of this package on the system dynamics can disrupt classical controllers, not originally pre-designed for this type of applications: particular control solutions must be applied.

Initiated for helicopters [BK09]; [BCHB10], the study and design of controllers adapted to load transportation has been investigated. Different control strategies have been explored for quadrotors, such as the application of sliding mode control [Zho+16], passivity-based control [Gue+15], adaptive control [DLB14] and nonlinear geometric control [ZS19]. Other methods focus on motion planning by generating trajectories that aim at minimizing the load's tracking errors [Zen+20]; [Foe+17]; [UE20]. Palunko, Fierro, and Cruz in [PFC12]; [PCF12] propose a solution based on dynamic programming which provides oscillation-free trajectory tracking. In a recent work [Yu+22], aggressive load-swing trajectories are obtained by having the constraints for load cable direction included into the trajectory generation via constraints on the load acceleration. Alternative works [Guo+20]; [ACC21] concentrate on modeling the payload as a disturbance and which is rejected by the control. For instance, in [ACC21] the load oscillations are attenuated using an extended high-gain observer for modeling payload disturbances.

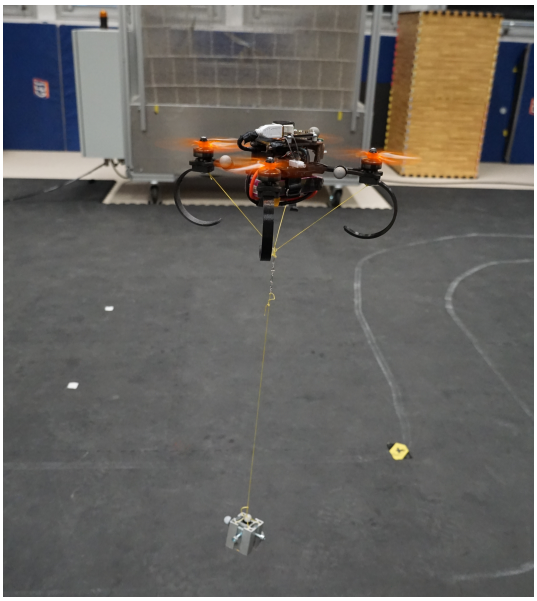


Figure V.1: An example of a suspended payload transportation using a quadrotor.

The aforecited research approaches frequently rely on the exact knowledge of the suspended load model and have shown good robustness to unmodeled dynamics. However, in addition to model errors that may appear during controller design phase, some assumptions on the system are often taken: known mass load, load movement in the plane, load linked with a rigid cable, the hang point at the center of mass of the UAV, etc. These assumptions over-simplifies the problem and can become problematic in concrete implementations of these methods. To solve this type of hypotheses, data-oriented methods have been tested in recent years to improve the trajectory generation, the modeling and the control using data flight sets. Solutions have been proposed to generate minimal oscillation trajectories by motion generation, as in [Fau+13] using learning or [LZH21]; [Hua+22]; [Fau+17] using reinforcement learning. For modeling and control purposes, one can cite [DLB14] which has designed a learning-based control scheme for piloting a quadrotor under coupling uncertainties. These learning methods, on the other hand, are extremely time consuming and require collecting a lot of relevant data to learn the controller. This process, often done by trial and error, can only be applied in ideal simulation environment and the controller implementation to real-world experimentation can be a hazardous procedure.

In the case of a controller designed over a nominal model without any modeling of the load itself, the latter will induce disturbances over control performance as an unmodeled dynamics. In the case of a model-based control taking into account the dynamics of the load itself, flight tracking performance is heavily dependent on the model reliability. Therefore, changing parameters would require a new tuning design effort to adapt the gains of the controller. Moreover, the modeling itself of the suspended payload is a risky task: in the later described application, the load is attached to the quadrotor frame by a four point string structure. To cite a few, payload and cable wind resistance or cable flexibility are also two huge obstacles to exhaustive modeling.

V.1.2 Proposed solution

In this chapter, we present a hybrid control method, combining an exact feedback linearization and a windowed deep neural network. It is used to perform quadrotor unknown payload transportation. The proposed control strategy is based on an exact linearization designed on a payload-free model of the quadrotor. All unknown dynamics, mainly generated by the payload during the mission, are first learned using a windowed DNN. It is next integrated in the exact linearization to correct the payload disturbed tracking. The proposed network is trained after collecting data, it is done offline in comparison to the previous chapter. It is then used to predict, in real-time, the disturbances generated by the additional load. It allows to correct the quadrotor tracking under the payload disturbances. In comparison to the previously mentioned learning methods, the DNN provides a fast learning and an easy implementation. The complete control scheme, using the proposed DNN, results in an exact linear behavior of the closed-loop disturbed system with improved trajectory tracking performance and payload oscillations attenuation.

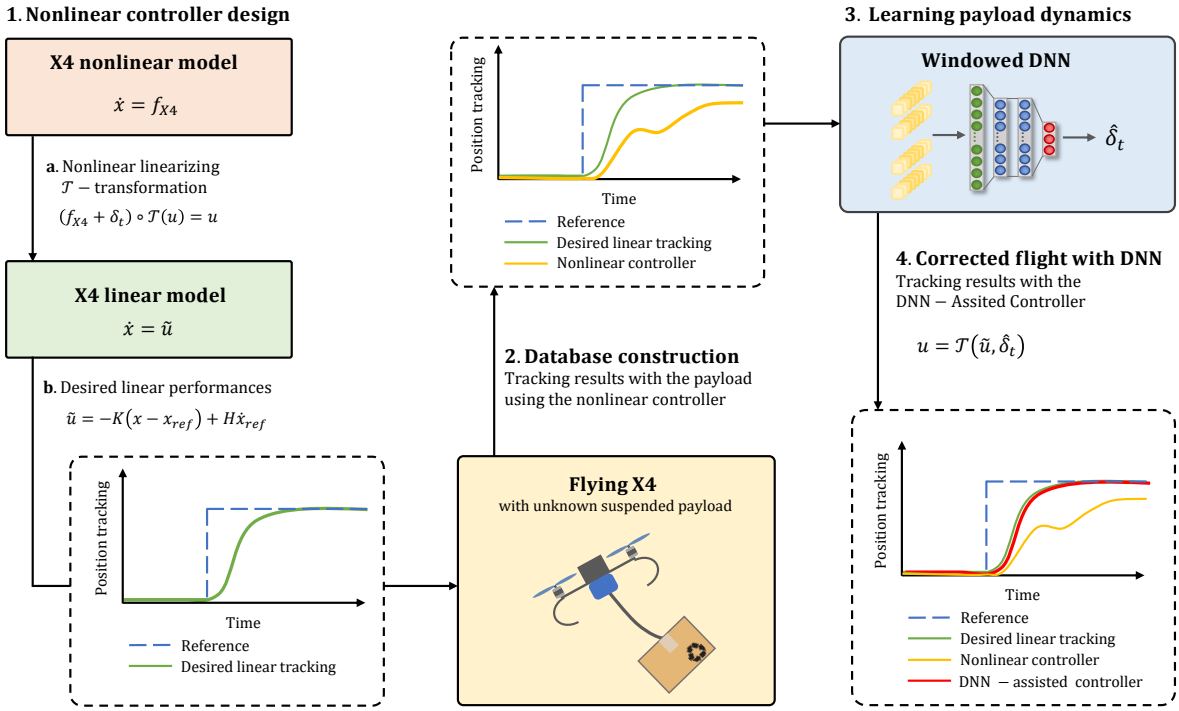


Figure V.2: The proposed methodology: DNN-assisted nonlinear feedback linearization for quadrotor unknown payload transportation.

The proposed methodology is summarized in Fig. V.2. It is composed of four main steps:

1. **Design of the initial nonlinear controller:** A nonlinear controller is firstly design to achieve trajectory tracking under payload transportation. It is an exact nonlinear feedback linearization. The desired performance is chosen on the exact linear model

obtained after the transformation.

2. **Database construction:** Using the previously designed nonlinear controller, one can perform unknown suspended payload transportation. A data collection is done to learn the payload dynamics in mission situation.

3. **Learning the unknown payload dynamics:** The database is used to learn the payload dynamics. A windowed deep neural network is created and trained (section III.3). After completing the learning, it is integrated to the nonlinear transformation.

4. **Flight using the DNN-based controller:** Finally, one can deploy the whole control scheme using the DNN-based correction controller. Flights are performed in tests cases scenarios (section III.5). The newly built controller allows to correct the tracking and fits the expect linear behavior from step n°1.

After quickly recalling the model used for control design in section V.2, the nonlinear controller is introduced in section V.3. The windowed DNN architecture is next presented in section V.4. Finally experimental payload transportation are carried out and analyzed in section V.5.

V.2 Reminder: quadrotor dynamics without payload

The main objective is here to perform the quadrotor trajectory tracking under payload transportation. The difficulty lies in the fact that we assume having no information about the payload used for the transportation task. In this manner, one has no knowledge on the cable length, rigidity nor the payload mass. Although modeling exists, the lack of knowledge of these parameters does not allow the model to be parameterized correctly and the controller to be accordingly adapted. Consequently, we chose to consider the disturbance generated by the payload as an unknown dynamic term for the quadrotor dynamics that will latter be learned. The model of the quadrotor used for the controller design is thus the standard model with a disturbance term essentially ruled by the payload. To design the controller, an exact feedback linearization, we will consider that the quadrotor, to which is attached the suspended payload, has the following model, adapted from the chapter I:

$$\begin{cases} \dot{\xi} &= v, \\ \dot{v} &= -g\vec{i_3} + \dfrac{1}{m}\boldsymbol{R}T\vec{b_3} + \delta_t, \\ \dot{\boldsymbol{R}} &= \boldsymbol{R}\boldsymbol{\Omega}_{\times}, \\ \dot{\Omega} &= -\boldsymbol{J}^{-1}\boldsymbol{\Omega}_{\times}\boldsymbol{J}\Omega + \boldsymbol{J}^{-1}\Gamma + \delta_r. \end{cases} \tag{V.1}$$

In (V.1), the additional terms corresponding to the unknown suspended payload disturbances are included via δ_t and δ_r. These terms also include non-modeled dynamics such as those presented in section I.4.3 as gyroscopic effect, blade flapping and also include non-modeled dynamics such as battery discharge and the unknown payload effect on the quadrotor dynam-

ics. They are denoted by δ_t and δ_r for respectively translational and rotational disturbance dynamics.

As for previous chapter, for experimental implementation purposes, it is assumed that the angular rate control loop is already implemented and has a sufficiently fast converging time. The dynamics considered in the following of the chapter is then given by (V.2):

$$\begin{cases} \dot{\xi} = v, & \text{(V.2a)} \\ \dot{v} = -g\vec{i_3} + \frac{1}{m}\boldsymbol{RT}\vec{b_3} + \delta_t, & \text{(V.2b)} \\ \dot{\zeta} = \boldsymbol{W}\Omega, & \text{(V.2c)} \end{cases}$$

To pilot the quadrotor with its suspended payload, we will use a nonlinear cascaded controller, using previously established model. Compared to chapters III and IV, this control will allow a more accurate tracking and will enable to get higher velocities than the linear cascaded controller. The overall control cascaded architecture remains unchanged: we still use position, velocity, attitude and angular rate control loops. We recall here the basis cascaded architecture was presented in section II.5.2.

V.3 The nonlinear controller

In this section, the nonlinear cascaded controller is presented. It aims at tracking a given trajectory reference for the quadrotor and its unknown suspended load. The global control architecture is two nonlinear cascaded for position& velocity and attitude controller. The angular rate control is still achieved with PIDs. The architecture is summarized in Fig. V.3. Here, we propose to use an exact linearizations as nonlinear control. They are performed on both position & velocity loops and on the angular loop.

After performing the linearization, a full state feedback using a linear quadratic regulator is used on the linear model obtained. The states used are estimated from the extended Kalman filter. Concerning the unknown suspended payload dynamics, they are learned via a deep neural network in section V.4. Using that estimation, a feedforward term is deduced to correct the exact linearization of the position & velocity loop, disturbed by the load.

V.3.1 Nonlinear attitude controller

An exact feedback linearization is used to achieve attitude control. We first recall here the equation that links Euler angles to angular rates and Euler:

$$\begin{bmatrix} \dot{\phi}, & \dot{\theta}, & \dot{\psi} \end{bmatrix}^T = \boldsymbol{W} \begin{bmatrix} p, & q, & r \end{bmatrix}^T \quad \text{(V.3)}$$

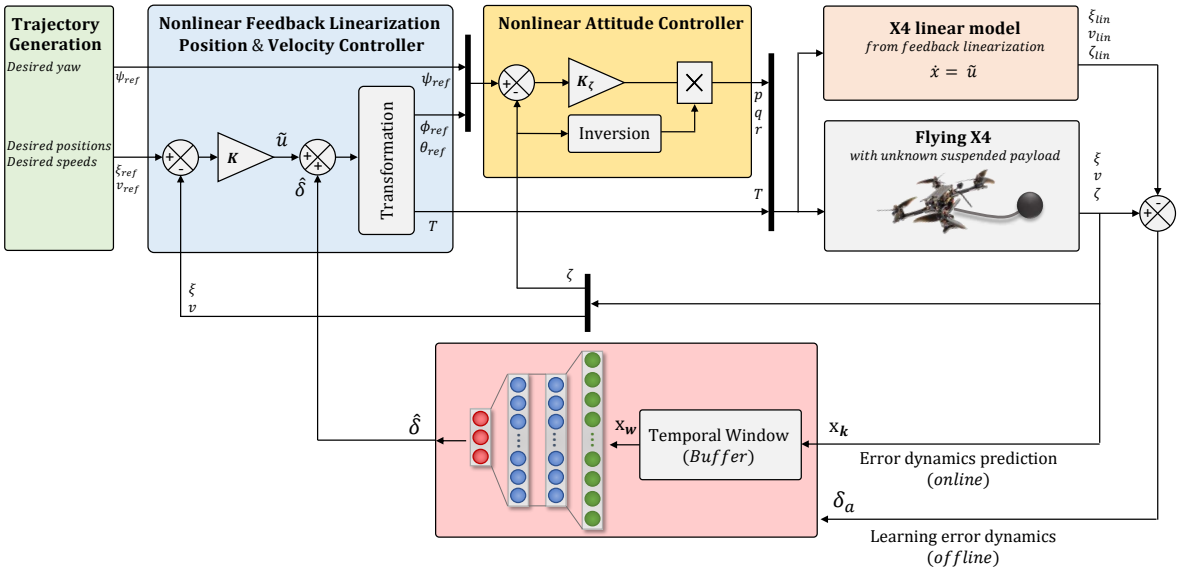


Figure V.3: The DNN-assisted nonlinear cascaded control to perform trajectory tracking with an unknown suspended payload.

The command variable is desired angular velocity denoted as $u_\Omega = \Omega = \begin{bmatrix} p, & q, & r \end{bmatrix}^T$. As the attitude can be estimated with a high precision using the EKF, one can compute the Wronskian matrix. That operation can be done sufficiently quickly to be performed online. One can then linearize and stabilize the loop, using the following control (V.4):

$$u_\Omega = -\boldsymbol{W}^{-1}(\zeta)\boldsymbol{K}_\zeta(\zeta - \zeta_{ref}), \tag{V.4}$$

where $\boldsymbol{K}_\zeta$ is a matrix gain tuned for tracking performance of the Euler angles ζ. Desired angle references, $\zeta_{ref} := u_\zeta$, is obtained by the position & velocity loop, presented in next section V.3.2.

V.3.2 Nonlinear position and velocity controller

We present here the position and velocity controller. It aims at reaching a given reference trajectory for the quadrotor. It provides to the inner loops the desired angles and thrust to achieve it. The proposed nonlinear controller is based on the exact linearization of the quadrotor.

First, one makes the assumption that $\psi \approx 0$, assuming the quadrotor is not performing yaw motion. It will be later achieve by a rotation of the computed commands. The second equation (V.2b) of the dynamical model of the quadrotor can then be simplified by (V.5), expressed on each axis:

$$\begin{cases} \ddot{x} &= \cos(\phi)\sin(\theta)\dfrac{T}{m} + \delta_t^x, \\ \ddot{y} &= -\sin(\phi)\dfrac{T}{m} + \delta_t^y, \\ \ddot{z} &= -g + \cos(\phi)\cos(\theta)\dfrac{T}{m} + \delta_t^z. \end{cases} \tag{V.5}$$

It must be noted that now $\delta_t = \begin{bmatrix}\delta_t^x, & \delta_t^y, & \delta_t^z\end{bmatrix}^T$ now includes nonlinear terms from $\psi \approx 0$ hypothesis to maintain the exact equivalence between (V.2b) and (V.5). To summarize quadrotor position & velocity model, we recall the following state variables as:

$$\xi := \begin{bmatrix}x, & y, & z\end{bmatrix}^T \quad \text{and} \quad v := \begin{bmatrix}v_x, & v_y, & v_z\end{bmatrix}^T.$$

The command variable is expressed as:

$$u_\zeta := \begin{bmatrix}\theta, & \phi, & T\end{bmatrix}^T.$$

The model used for the controller design can be written as the following state space representation:

$$\begin{cases} \dot{\xi} &= v, \\ \dot{v} &= f_{X4}(u_\zeta) + \delta_t, \end{cases} \tag{V.6}$$

where:

$$f_{X4}(u_\zeta) = \begin{pmatrix} f_{X4}^x \\ f_{X4}^y \\ f_{X4}^z \end{pmatrix} = \begin{pmatrix} \cos(\phi)\sin(\theta)\dfrac{T}{m} \\ -\sin(\phi)\dfrac{T}{m} \\ -g + \cos(\phi)\cos(\theta)\dfrac{T}{m} \end{pmatrix}. \tag{V.7}$$

The model given by (V.6) is composed of nonlinear known function f_{X4} dependent of the chosen command and a nonlinear unknown function δ_t mainly ruled by the unknown payload dynamics.

V.3.2.1 Exact linearization principle

For control purpose, we propose to exactly linearize the model (V.6), that includes the unknown suspended payload. Here we rely on the work of [Voo09] to establish the linearization. In a first step, we assume the complete knowledge of δ_t. In the following section V.4, explanations concerning the learning of this disturbance term is given. Here, we are defining $\mathcal{T}$-transformation such that:

$$(f_{X4} + \delta_t) \circ \mathcal{T}(u) = u. \tag{V.8}$$

By taking input $u_\zeta = \mathcal{T}(\tilde{u}_\zeta)$, the system (V.6) can be written in a linear form:

$$\begin{cases} \dot{\xi} &= v, \\ \dot{v} &= \tilde{u}_\zeta. \end{cases} \tag{V.9}$$

It is then easy to design a controller for this linear system (V.9) using standard control tools.

V.3.2.2 Exact linearization derivation

Before performing the linearization, let us define the three following input variables $\tilde{u}_x$, $\tilde{u}_y$ and $\tilde{u}_z$ such as:

$$\begin{cases} \dot{v}_x &= \tilde{u}_x &= f_{X4}^x(\phi, \theta, T, \delta_x) + \delta_t^x, \\ \dot{v}_y &= \tilde{u}_y &= f_{X4}^y(\phi, T, \delta_y) + \delta_t^y, \\ \dot{v}_z &= \tilde{u}_z &= f_{X4}^z(\phi, \theta, T, \delta_z) + \delta_t^z. \end{cases} \tag{V.10}$$

By defining, $\alpha = \sin(\phi)$ and $\beta = \sin(\theta)$, the following properties can be used:

$$\cos(\phi) = \pm\sqrt{1-\alpha^2} \quad \text{and} \quad \cos(\theta) = \pm\sqrt{1-\beta^2} \tag{V.11}$$

Then, substituting (V.11) in (V.10), on obtains the following system of equations:

$$\begin{cases} \tilde{u}_x &= \pm\sqrt{1-\alpha^2}\beta\dfrac{T}{m} + \delta_t^x, \\ \tilde{u}_y &= -\alpha\dfrac{T}{m} + \delta_t^y, \\ \tilde{u}_z &= -g \pm \sqrt{1-\alpha^2}\sqrt{1-\beta^2}\dfrac{T}{m} + \delta_t^z. \end{cases} \tag{V.12}$$

We now isolate T, α and β variables from (V.12). To do so, we first assume that $\tilde{u}_x \neq \delta_t^x$, we obtain:

$$\begin{cases} \beta &= \pm\left[\dfrac{(\tilde{u}_z + g - \delta_t^z)^2}{\tilde{u}_x - \delta_t^x} + 1\right]^{-1/2}, \\ T &= \pm m\sqrt{\dfrac{(\tilde{u}_x - \delta_t^x)^2}{\beta^2} + (\tilde{u}_y - \delta_t^y)^2}, \\ \alpha &= -(\tilde{u}_y - \delta_t^y)\dfrac{m}{T}. \end{cases} \tag{V.13}$$

Now, we need to define the sign of these variables in (V.13). First, thrust is always positive by definition. Knowing that ϕ is defined in the range $[-\pi/2; \pi/2]$ and using equations (V.5), we deduce that θ has the same sign as $\tilde{u}_x - \delta_t^x$. We can then define the exact linearizing $\mathcal{T}$-transformation by $u_\zeta = \left[\phi, \theta, T\right]^T = \mathcal{T}(\tilde{u}_x, \tilde{u}_y, \tilde{u}_z)$. We have to distinguish several cases according to the values of input variables $\tilde{u}_x$, $\tilde{u}_y$ and $\tilde{u}_z$.

For $\tilde{u}_t^x \neq \delta_x$, we obtain the following expressions:

$$\begin{cases} T &= m\sqrt{\dfrac{(\tilde{u}_x - \delta_t^x)^2}{\beta^2} + (\tilde{u}_y - \delta_t^y)^2}, \\ \phi &= \arcsin\left(-(\tilde{u}_y - \delta_t^y)\dfrac{m}{T}\right), \\ \theta &= \text{sign}(\tilde{u}_x - \delta_t^x)\arcsin\left(\pm\left[\dfrac{(\tilde{u}_z + g - \delta_t^z)^2}{\tilde{u}_x - \delta_t^x} + 1\right]^{-1/2}\right). \end{cases} \tag{V.14}$$

For $\tilde{u}_x = \delta_t^x$, we have four cases:

1. If $\tilde{u}_y = \delta_t^y$ and $\tilde{u}_z = \delta_t^z$: the quadrotor is not moving nor it is hovering. We then just need to compensate for the gravity by using following input $\Big[T, \phi, \theta\Big] = \Big[mg, 0, 0\Big]$.

2. If $\tilde{u}_y \neq \delta_t^y$ and $\tilde{u}_z \neq \delta_t^z$: the quadrotor is not moving along x axis. We thus need to maintain $\theta = 0$, to avoid pitching. The transformation is then given by:

$$\begin{cases} T &= m\sqrt{(\tilde{u}_z + g - \delta_t^z)^2 + (\tilde{u}_y - \delta_t^y)^2}, \\ \phi &= \arcsin\left(-(\tilde{u}_y - \delta_t^y)\dfrac{m}{T}\right), \\ \theta &= 0. \end{cases}$$

3. If $\tilde{u}_y = \delta_t^y$ and $\tilde{u}_z \neq \delta_t^z$: the quadrotor is only moving along z axis. Both angles need to be maintained at 0, and the only variable to be controlled is the thrust, using following expression:

$$\begin{cases} T &= m\sqrt{(\tilde{u}_z + g - \delta_t^z)^2}, \\ \phi &= 0, \\ \theta &= 0. \end{cases}$$

4. If $\tilde{u}_y \neq \delta_t^y$ and $\tilde{u}_z = \delta_t^z$. This case is obtained when $\phi = 90°$. It is experimentally avoided using an adapted trajectory generation, not requiring high angles to move the quadrotor. It is also prevented using saturation on Euler angles.

We now have the complete expression of the $\mathcal{T}$-transformation that allows us to exactly linearize the nonlinear position & velocity system. We can focus on the control of the obtained linear model.

V.3.2.3 Control of the exact linearized system

The system {quadrotor+load}, after applying the $\mathcal{T}$-transformation, is now given by linear state space representation (V.9). To control, this linear system, several solutions exist. Here,

we have decided to design a state feedback using a linear quadratic regulator, as we have good estimate of the full state. The selected controller is given by (V.15):

$$\tilde{u} = -\boldsymbol{K_\xi}(\xi - \xi_{ref}) - \boldsymbol{K_v}(v - v_{ref}) + \dot{v}_{ref} \tag{V.15}$$

where the command $\tilde{u} = \begin{bmatrix}\tilde{u}_x, & \tilde{u}_x, & \tilde{u}_x\end{bmatrix}^T$ and matrices gains $\boldsymbol{K}_\xi$ and $\boldsymbol{K}_v$ are chosen by minimizing the following quadratic cost (V.16):

$$J = \int_0^{+\infty} [(\xi - \xi_{ref})^T \boldsymbol{Q_\xi} (\xi - \xi_{ref}) + (v - v_{ref})^T \boldsymbol{Q_v} (v - v_{ref}) + \tilde{u}_\zeta^T(t) \boldsymbol{R} \tilde{u}_\zeta(t)] dt \tag{V.16}$$

The weight matrices denoted $\boldsymbol{Q_\xi}$, $\boldsymbol{Q_v}$ and $\boldsymbol{R}$ are tuned to get the desired tracking performance making compromise between state convergence and command usage. Here, they are chosen empirically to a step response in each axes. We desire a fast response while avoiding overshoots and oscillations. To tune the gains it is first performed in simulation. Next, we fine-tuned manually these gains with experimental tests. No suspended payload is used to set and tune the gains.

V.3.2.4 Summarizing

The exact linearization of the nonlinear quadrotor system is achieved via the $\mathcal{T}$-transformation $u_\zeta = [\theta,\ \phi,\ T]^T = \mathcal{T}(\tilde{u}_x, \tilde{u}_y, \tilde{u}_z, \delta_t^x, \delta_t^y, \delta_t^z)$. It allows to control a linear model and achieve tracking in position for the quadrotor with its suspended payload. The command u_ζ is then given as a reference for the previously established attitude control loop.

Remembering the assumption of the knowledge of all non-linearities, it is necessary to get the δ_t to correctly perform the linearization. It is consequently needed to get information on disturbances generated by the payload. It exists several methods to estimate such disturbances, for instance high-gain observers can be used. However, that type of observer generally needs a model of the disturbance to estimate it. It might by a long and complicated process. To avoid tuning complexity and correctly learn a controller without any information about the payload, we choose to use a deep neural network, as in previous chapters. Its architecture and training are given in the next section V.4.

V.4 Neural payload estimation

In this section, we explain how error dynamic δ_t, including the suspended unknown load disturbances, is learned using a windowed deep neural network. It will next be used to estimate δ_t in real-time and allow the exact linearization for the position & velocity control loop V.3. The neural network training and validation are performed offline after collecting a sufficiently large amount of flight data. The DNN is trained to predict $\delta_t = \dot{v} - f_{X4}$, where $\dot{v}$ is the vector of measured accelerations.

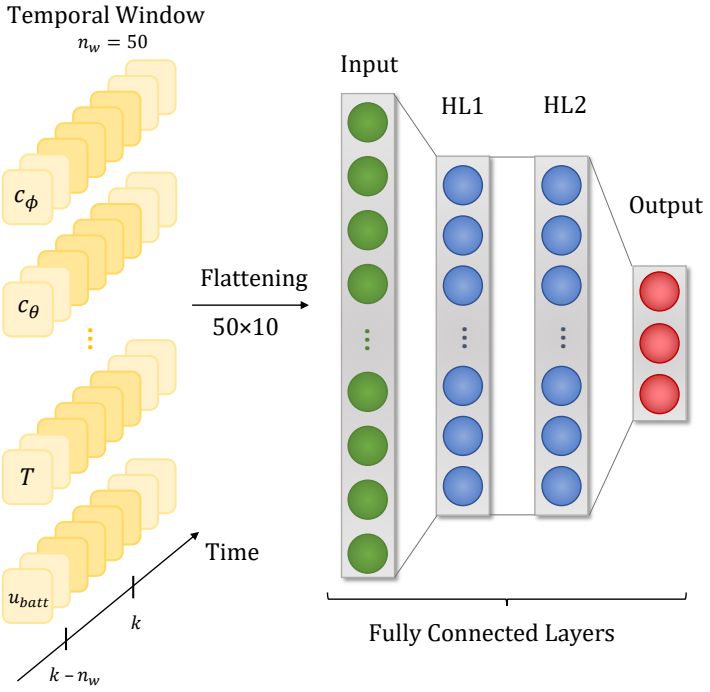


Figure V.4: The windowed deep neural network architecture.

The chosen network is a deep feed-forward network, composed of two hidden layers of 64 units, with ReLU function as activation. The DNN input corresponds to a windowed temporal sequence containing $n_w = 50$ samples of past selected data inputs.

The windowed deep neural network is denoted by $\hat{\delta}_t$ and is given by (V.17), see Fig. V.4:

$$\hat{\delta}_t(\mathrm{x}_w, \vartheta) = \mathbf{W_3}^T h(\mathbf{W_2}^T h(\mathbf{W_1}\mathrm{x}_w + \mathbf{B_1}) + \mathbf{B_2}) + \mathbf{B_3}, \tag{V.17}$$

where following notations are used:

- ϑ is the vector composed of weights and biases of the different layers:

$$\vartheta = \{\mathbf{W_1} \quad \mathbf{W_2} \quad \mathbf{W_3} \quad \mathbf{B_1} \quad \mathbf{B_2} \quad \mathbf{B_3}\}$$

- x_w is a vector resulting from the concatenation of a time window on the selected inputs x_k of the network, given in (V.18). The temporal length of that window is denoted as n_w.

$$\mathrm{x}_w = \left[\mathrm{x}_1^T, \quad \mathrm{x}_2^T, \ \ldots\ , \quad \mathrm{x}_{n_w-1}^T, \quad \mathrm{x}_{n_w}^T\right]^T.$$

- h is the element-wise rectified linear unit (ReLU) activation function. It is used for the hidden layers. We recall its definition as: $h(\cdot) = max(\cdot, 0)$.

The windowed inputs x_k are selected based on the dynamics of the quadrotor and from previous chapter experience (III and IV). We still provide sines and cosines of angles, thrust

command, battery tension and squared velocities. No further information is provided, therefore the DNN has no additional information about the unknown payload.

$$\mathrm{x}_k = \Big[c_\phi, \quad s_\phi, \quad c_\theta, \quad s_\theta, \quad v_x^2, \quad v_y^2, \quad v_z^2, \quad u_{batt}, \quad T, \quad z\Big]^T \tag{V.18}$$

For that application, we decided to work using a windowed input for the deep neural network. Other solutions using recurrent neural networks or long short-term memory [She20] exist and we considered them to model the disturbance. However, several choices have motivated the choice of the windowed solution:

1. The use a simple structure that we understand,
2. Some preliminaries tests have shown equivalent results in terms of modeling,
3. As already mentioned, DNN are able to learn such disturbances, the window is here to help and improve the learning in a simple way.

The loss function used for the training process is the Mean Squared Error (MSE) between true measured values δ_t and predictions $\hat{\delta}_t$ (V.19):

$$\min_{\vartheta} \mathcal{L}(\vartheta) = \min_{\vartheta} \frac{1}{N} \sum_{k=1}^{N} \|\delta_t - \hat{\delta}_t\|^2, \tag{V.19}$$

The optimizer solver used is Nesterov-Accelerated Adaptive Moment optimizer (see B.3.2, algorithm 2). The hyper-parameters used for the training and the information on the DNN architecture are summarized in Table V.1. The values of the training parameters are determined through practical experimentation to ensure efficient and satisfactory convergence of the DNN. The window length given by n_w is chosen based on the frequency of oscillations caused by the unknown payload during a flight and represents half of an oscillation period. A brief study of the variation of the number of units n_u and layers n_l led us to the selection of these parameters.

The DNN training is here performed offline using a previously created database of payload transportation flights. There were done using the nonlinear controller described in section V.3 and considering no error disturbances $\delta_t = \mathbb{0}_3^T$ for the $\mathcal{T}$-transformation. The training set is covering 45 minutes of data and validation set is about a 15 minutes. It includes different starting positions of the payload around the quadrotor, see Fig. V.5. Initial conditions allows to generate differently oriented oscillations at takeoff. The database is composed of different motions in space at different velocities. It includes steps in different directions, circles, spiral motions, etc. Takeoff and landing phases are also added to the database.
The effectiveness of the learning is validated using the exact linearization controller with the DNN estimation for the transformation. The results are experimentally obtained and presented in next section V.5.

In the following, the nonlinear controller using DNN estimation is referred as DNN-assisted feedback linearizing cascaded controller.

Parameter	Value
Number of inputs	10
Number of hidden layers n_l	2
Number of units n_u	64
Window size n_w	50
Total input size	500
Batch	256
Epoch	800
Learning rate l_r	0.002
μ	0.9
ν	0.9999

Table V.1: Parameters selected for the windowed DNN and hyperparamters for its learning process.

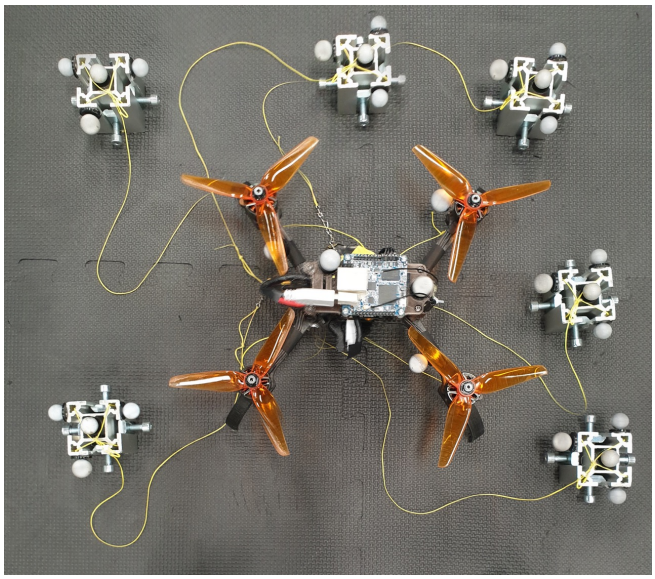


Figure V.5: Initial load position in database construction.

V.5 Experimental results

To evaluate the performance of the proposed nonlinear neural controller, we conduct payload transportation experimentation. Most of the experimental setup remains unchanged compared to the previous chapters. It should be recalled that the chapter II provides a lot of information about MOCA room, software, etc. used in the experiment.

The unknown payload is connected to the quadrotor via a non-rigid wire, as it can be noted in Fig. V.1. The payload mass is 180g. Four reflectors are attached to the payload to track its spatial position.

In the following, we start by validating the nonlinear neural controller in comparison to the initial nonlinear controller using integral action in sub-section V.5.1. It is next compared to the initial controller using a high-gain observer in sub-section V.5.2.

V.5.1 Baseline controller comparison

Real-world experiments are conducted to confirm the effectiveness of the proposed DNN-assisted feedback linearizing cascaded controller.

First, experimental tests without the DNN estimation are performed to collect payload transportation data and learn δ_t as explained in the dedicated section V.4. After completing offline training, the model is used online for predictions.

The proposed controller is tested on an unseen transportation scenario, referred to as the test scenario. This proposed scenario includes:

- a take-off phase,
- successive steps along x, y and z directions,
- two circular motions,
- and a landing procedure.

Here, we performed twice this experimentation: one using the DNN-assisted feedback linearizing cascaded controller and a second time using the nonlinear cascaded controller with an integral action but without a disturbance observer. It is done to first assess the improvement of the DNN payload estimation. It is important to note that the initial conditions for the quadrotor and the payload are the same in both scenarios.

The obtained position tracking results over the test scenario are presented in Fig. V.6. The three plots represent quadrotor x, y and z positions over the time. The green dashed-line represents the expected linear behavior of the quadrotor without any disturbance. It represents the behavior obtained after the exact linearization without disturbances generated by the payload and the other neglected effects: the battery discharge, etc. The closer the controller used is to this value, the better the tracking performance.

It is clear that both controllers are able to successfully complete the task. However, as expected, the controller without DNN estimate has two drawbacks. First, it exhibits a slower response time, which is particularly noticeable during step tracking phases in all three axes. Second, it presents small oscillations over x and y axes. These oscillations are mainly due to the unexpected movements of the load during acceleration phases. Since the disturbances are not constants, the nonlinear controller shows poor performance.

On the other hand, the DNN controller effectively cancels or minimizes these undesirable oscillations in all axes. This is made possible by the DNN, which takes into account the oscillations generated by the payload and applies corrections to the linearization of the model.

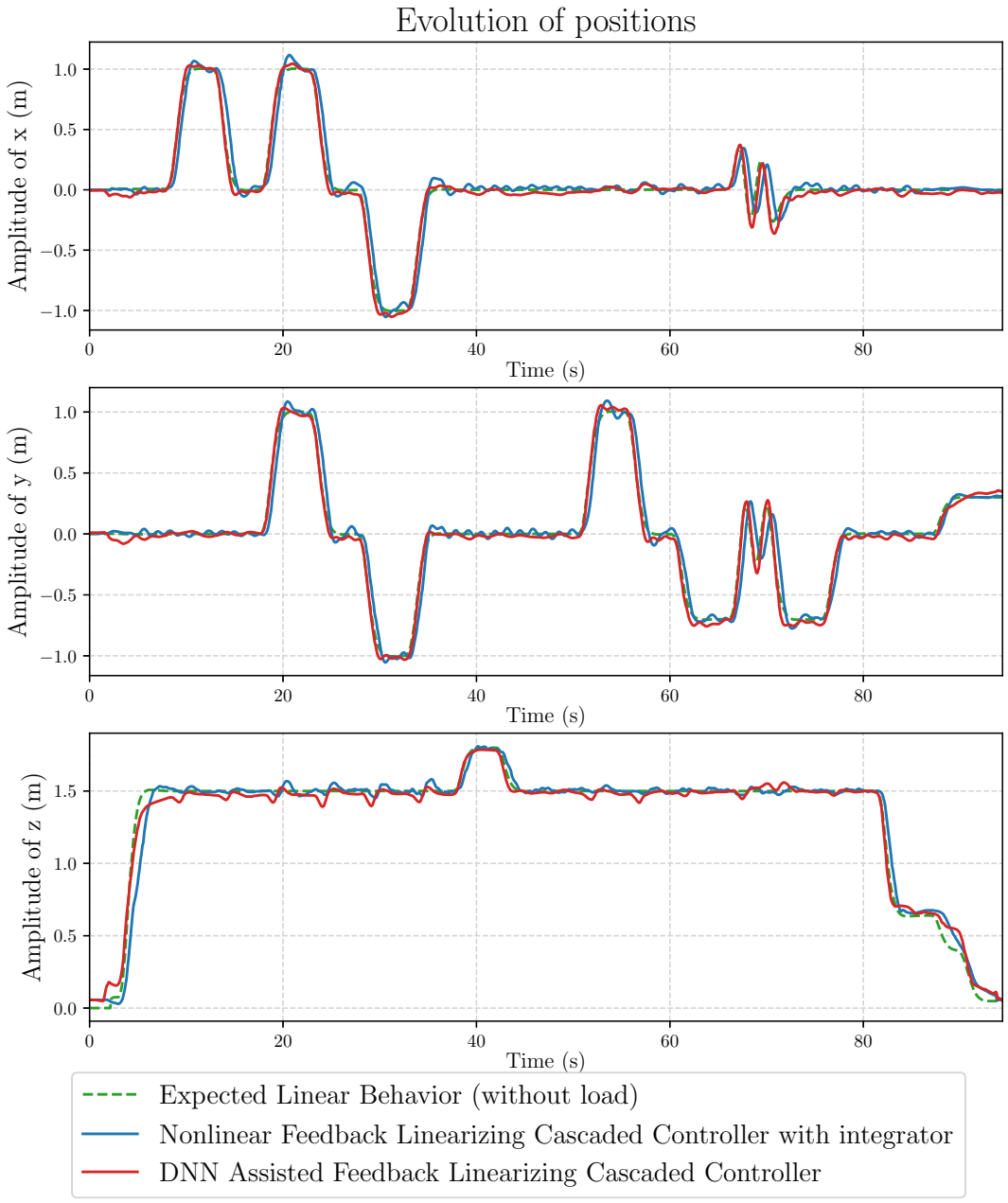


Figure V.6: The spatial position of the quadrotor with the unknown suspended payload over the experimental test scenario.

A way to evaluate performance of such approach is to compute the root mean squared error between the expected linear behavior without load and the obtained results considering both controllers. The RMSEs computed over the transportation scenario are given in Table V.2. One can notice a factor two between the controllers. It assesses the great interest of using the DNN-assisted feedback linearizing cascaded controller.

Another way to assess the impact of using the DNN-assisted controller is to examine the

	Baseline controller (with integrator)	DNN-assisted controller
RMSE x-axis	0.09 m	0.045 m
RMSE y-axis	0.098 m	0.050 m
RMSE z-axis	0.077 m	0.046 m

Table V.2: Root mean squared errors on each axes for transportation scenario, computed for both controllers with respect to the expected linear behavior.

temporal evolution of the payload positions. Indeed, reflectors placed on the payload allow us to get this information. The spatial position in x and y axes are displayed in Fig. V.7 on the given test scenario for both controllers.

The payload is strongly oscillating during the scenario with the nonlinear controller without DNN assistance. Whereas in the DNN assisted scenario the fluctuations are reduced. Indeed, a better tracking in position for the quadrotor implies a less oscillating scenario for the payload. The oscillations of the payload, with a frequency at 0.8Hz, are attenuated by more than 20dB with the DNN controller. The Discrete Fourier Transform (DFT) of the signals around load additional oscillations are presented in Fig. V.7 and confirm this observation.

V.5.2 Controller with high-gain observer comparison

This section present the comparison of the proposed approach compared to the nonlinear controller with a high-gain observer used as payload disturbance observer, such as presented in the work [ACC21]. In practice, the disturbance observation from the high-gain is injected in the nonlinear controller presented before in section V.3.1. More details about high-gain observers is given in the appendix A.

The results on the same payload transportation test scenario are given in Fig. V.8. Initial conditions are kept identical. Both controllers demonstrate improved tracking performance compared to the baseline controller. Indeed, for both controller, oscillation reduction can be noted, particularly along x and y axes. The controller with the high-gain observer is slower in x and y axes compared to the DNN-assisted one. Some overshoots during the circle phase can also be observed.

As for previous comparison, the load positions can be analyzed on the same transportation scenario. They are represented in Fig. V.9. The controller with the high-gain observer still presents the payload oscillations, with a high amplitude during acceleration phases. These oscillations take more time to be attenuated at hover. Here, one can assess that the DNN solution better models the payload dynamics in comparison to the observer. It thus allows a better disturbance rejection. The DFT of the signals around payload additional oscillations supports this observation.

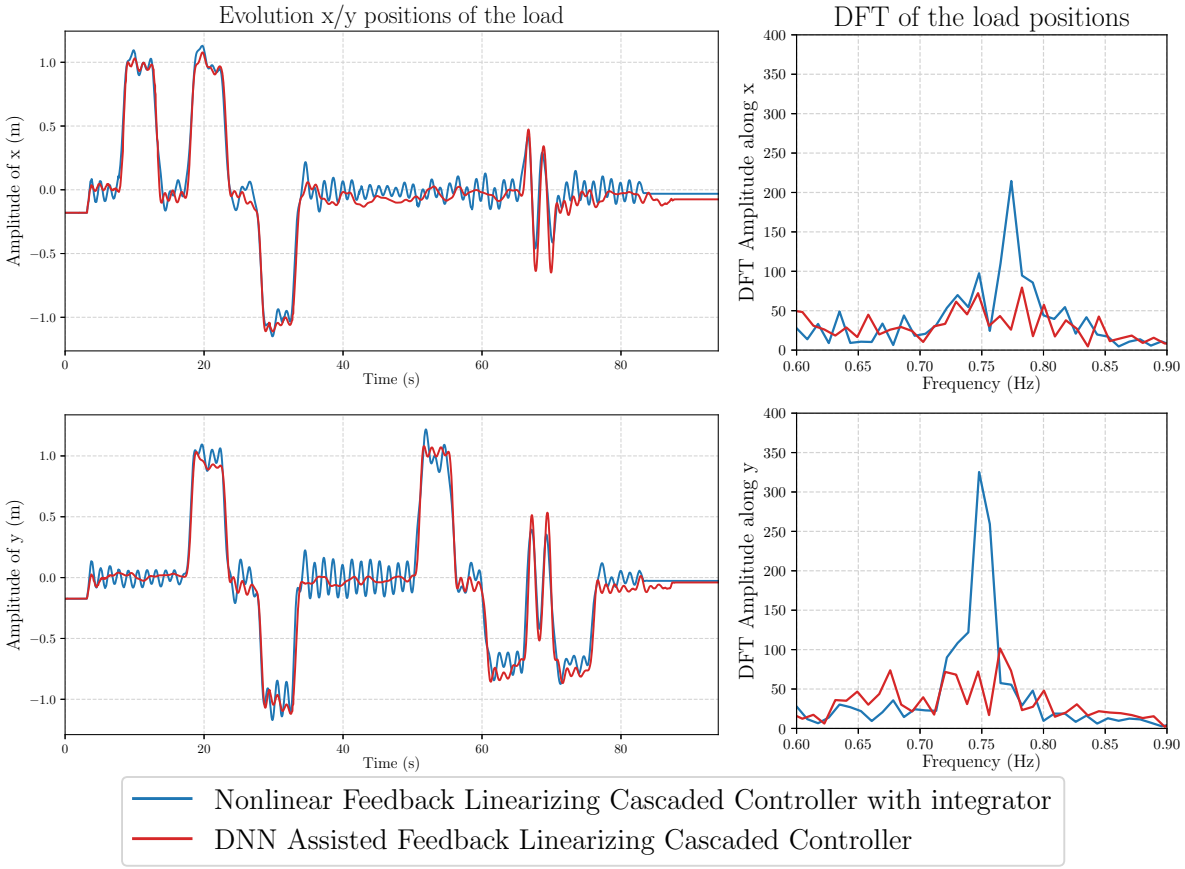


Figure V.7: Load positions during transportation test flight and DFT (modulus) of the signals around load oscillations.

Another aspect that can be highlighted between the two methods is the whole tuning time. It is often forgotten to specify the time that can take the implementation and tuning of such methods. In that particular case, finding efficient gains for the high-gain observer is a long and complicated process. Indeed, simulations are essential to find first acceptable observer gains. Assuming we have a rather accurate quadrotor with a suspended payload simulator, which is not the case here. Then, these gains must be refined by carrying out several experimentation with the payload. To summarize, it is needed a model of the disturbance, a simulator of the whole system, a first gain set and next to refine obtain gains in experiment. It takes a lot of time to be able to correctly flight with a payload. On the other hand, the proposed approach with the windowed deep neural learning requires flights with the baseline controller, which is easy to design. The complexity is reduced to the selection of hyper-parameters for the DNN. After that, the DNN-assisted controller can be directly used on the scenario.

V.5.3 Discussion on learning

In the following, we provide some discussions concerning the DNN training and generalization.

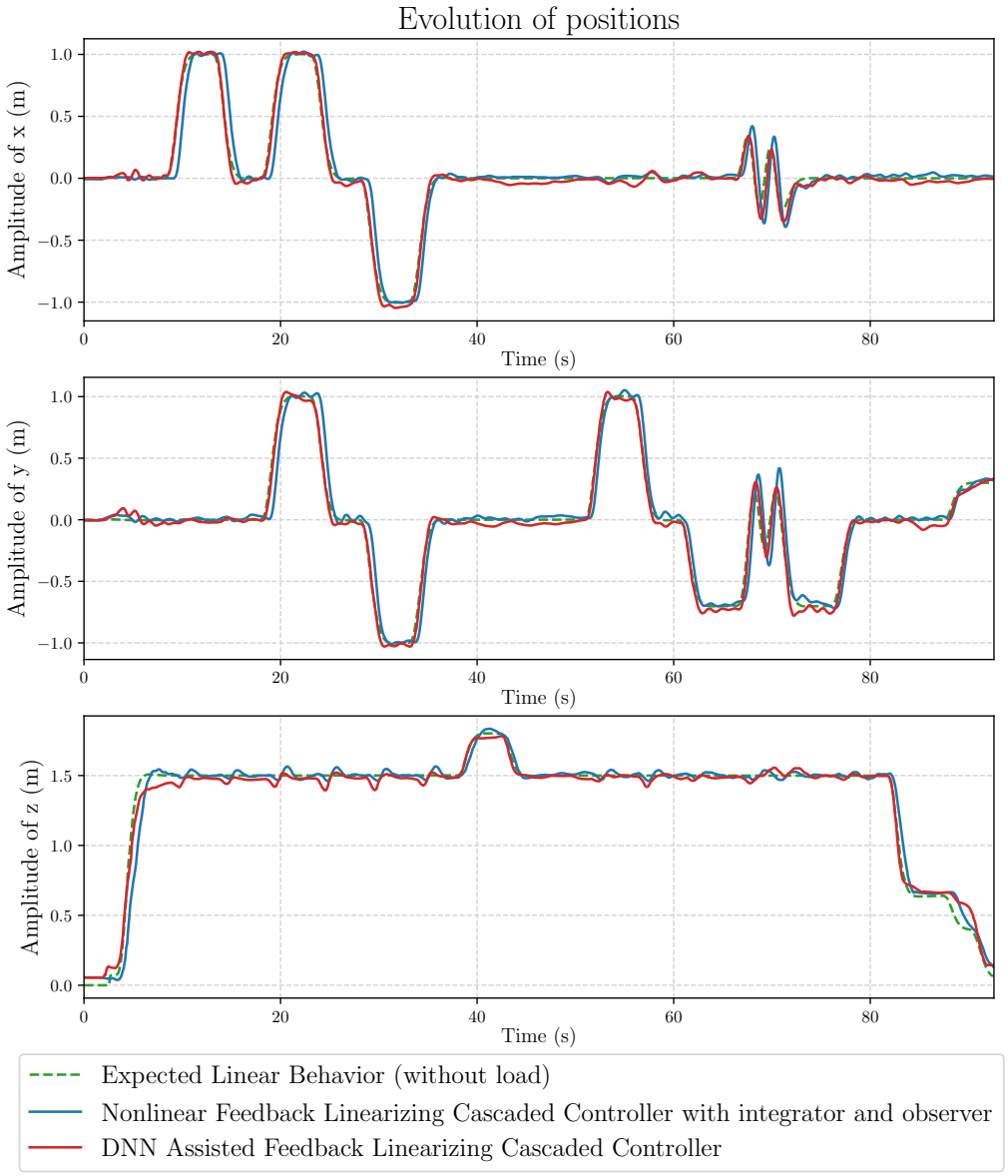


Figure V.8: The position of the quadrotor with the unknown suspended payload over the test scenario. It is a comparison between the nonlinear controller using a high-gain observer and the DNN-assisted feedback linearizing cascaded controller.

The training is done over a relatively small dataset compared to classical databases uses for deep learning in vision, for instance. However, it contains enough data considering this regression application: learning the payload dynamics on a given environment. It is composed of basics movements such as circles and steps at different velocities. The learning is done in a way that it does not over-fit the data set using validation set for model selection. The selected deep neural network is finally tested it the presented test case that is not present in the given database. However, as motions are similar (steps or circle) it is achieved with various amplitudes and in a various order. Selected scenario velocities are done between minimum

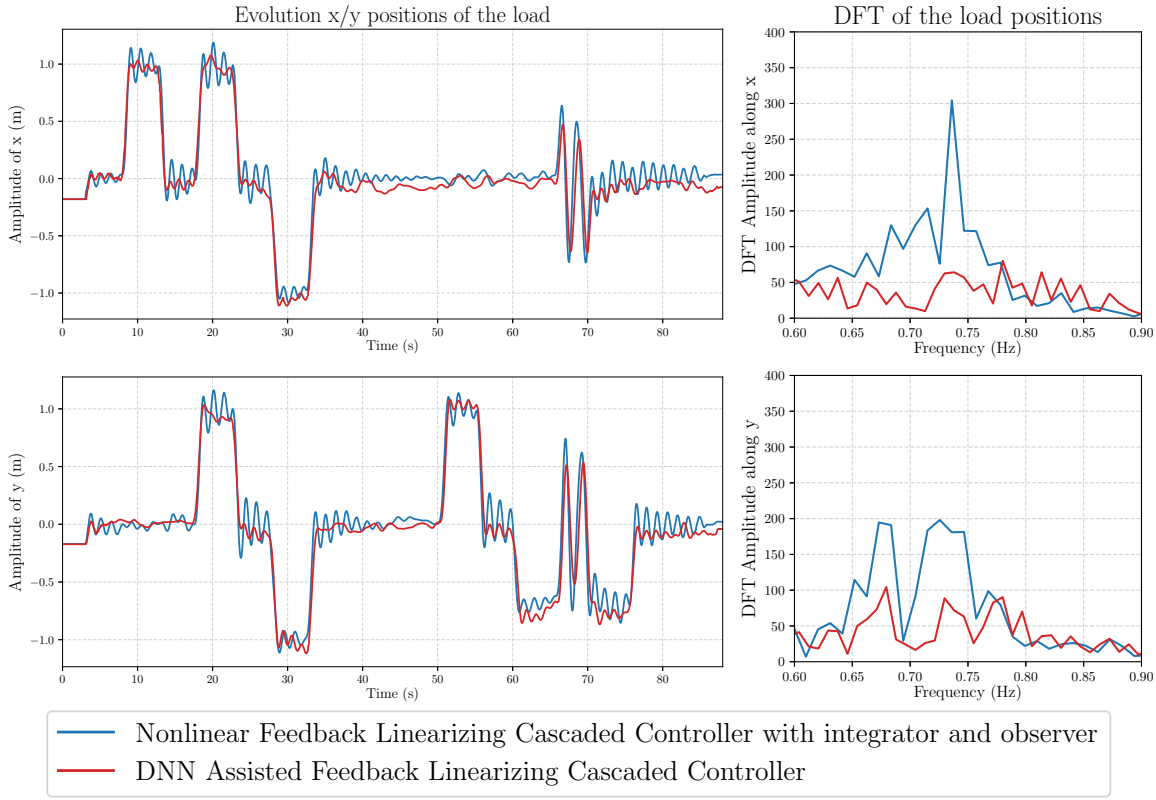


Figure V.9: Experimental payload positions during transportation test flight and DFT (modulus) of the signals around load oscillations.

and maximum trajectory speed on the database.

Testing the DNN-assisted controller in a completely different scenario, for instance in another environment presenting wind, without including it in the learning database is a nonsense that will fatally lead the DNN-assisted to provide significant errors and potentially led the quadrotor to a crash. In the same way, the presented work is carried out with an unknown payload and a non-rigid wire length. To better generalize over several payload weights or cable lengths, data should be added to the training database considering those parameters (that could either be included as input to the DNN or not). The DNN should be able to generalize over those parameters, and it constitutes a way to enhance the proposed method.

A video of unknown payload transportation using proposed controller is available via the following link: `youtu.be/MtCwAlrSIxo`

V.6 Conclusions and perspectives

Transporting a suspended load with a quadrotor is a challenging task. Indeed, the disturbances caused by the oscillating payload during maneuvers can default the controller. The design of an efficient autopilot is a tedious process as it may require an accurate modeling of the payload and is dependent on the mechanical characteristics of the payload: cable, weight, etc. Using an observer to estimate the unknown dynamics is a useful and common technique for estimating disturbances. However, in practice, synthesizing an observer, for this type of very complex nonlinear system can be hard and may take a long time before achieving satisfactory results.

The proposed solution to handle the problem and reduce the implementation complexity, is a deep neural learning of the payload dynamics. By performing flights with an initial satisfactory controller, one can learn the disturbances generated by the suspended payload. Proposed neural structure is a windowed DNN, to well predict the behavior. Afterwards, the learned dynamics can be introduced in the nonlinear transformation to achieve a better trajectory tracking. One advantage of this method is that it does not require an explicit model of the payload dynamics. The learning is done through collected data. The obtained results demonstrate an improved tracking of the initial controller under payload disturbances. It also results in a diminution of the oscillation of the payload during the flight. This side effect is even more beneficial as it reduces the potential damage of the payload. It also presents a more desirable behavior than that of a linearizing control with an integral action and high-gain observer.

A logical extension of this work would be to implement the learning process online. Indeed, it will avoid landing before applying the DNN estimation to the linearizing transformation. A possible way to perform could be inspired from the previous chapter. As previously discussed, another extension of this work should be the management of a broader set of different cable lengths and payload weights within a unique learning structure. Both the database and the structure of the DNN should be refined to address this issue.

Conclusion and perspectives

V.7 General overview

This thesis was carried out in a research team which develops algorithms for automatic piloting of quadrotors. The objective is to propose efficient flight algorithms in mission situations. Here, we focus on trajectory tracking accuracy. The developed algorithms must be efficient to rapidly respond to unforeseen events such as the appearance of external disturbances, like wind gusts. These algorithms must be efficient and easily implementable in order to be deployed in a large variety of quadrotor models.

The main motivation of this work is to bring together two worlds: control and learning in order to improve the flights tracking performance of a quadrotor. This thesis brings contributions in various forms, indeed both theoretical and methodological contributions are proposed. In addition, it also include a large experimental contribution, since many efforts have been made to ensure that all proposed solutions work in practice in the GIPSA-lab experimental room.

We proposed a first architecture to combine learning and a standard control approach in chapter III. It is based on the frequently employed cascaded architecture for piloting quadrotors. It is easy to tune as it is based on cascaded loops, independently built. The proposed solution is a DNN-based controller in position loop. That DNN is learned using flights data from previously performed missions, including as much possible movement of the quadrotor in different situations. After the offline learning of the DNN, it is deployed in the position control loop. It acts by correcting the command input to get closer to the initial desired linear behavior. It results in a better tracking of the quadrotor position under internal model errors but also with external disturbances. This approach is firstly validated in simulation environment such as Gazebo. It is next tested in real-world flights using a motion capture room. The advantage of this method is that the flight of the quadrotor is ensured by the standard cascaded control architecture and is enhanced by the learning capability of deep neural networks. This method suffers two principal disadvantages. The first one is the offline learning, requiring first to perform a considerable number of flights with the initial controller, in a degraded mode, before getting a more precise and reactive tracking. The second one is the stability limited to experiments close to the firstly built database.

To address the two major concerns of the previous approach, we proposed a novel solution. This second algorithm is an event-based neural learning strategy. It is based on the previous approach keeping the cascaded architecture of the quadrotor. It learns a deep neural network iteratively based on two criteria. A first criterion which ensures the stability of the closed-loop system with the deep neural network correction. A second criterion focuses on the tracking performance during flight. The events are based on these two criteria. The strategy switches to the initial controller, in case of instability, or re-learn the DNN, in case of inaccurate

learning. The re-learning is based on a new data collection and consists in an improvement of the previously found network weights. This strategy demonstrates a good effectiveness in both simulation and experimental test, even under external disturbances.

Finally, we have studied another autopilot algorithm that focuses on the transportation of an unknown suspended payload. The developed algorithm is based on a nonlinear control coupled with a windowed deep neural network. Indeed, the load transport being a more delicate problem, a more precise and efficient initial algorithm is necessary before performing tracking with the unknown payload. The nonlinear controller allows to have a more reactive control than the linear control previously proposed. It allows to realize an exact linearization of the system by using the learning of the window deep neural network. It is used to predict the errors generated by the unknown suspended payload by learning it from previous flights. The coupling of the nonlinear control and the windowed deep neural network allows to correct the trajectory and to reduce the oscillations generated during flight.

All the proposed solutions are combining the advantages of both standard controller and learning methods. The standard control solution is used to bring a first basis for the quadrotor auto-piloting in a secured manner from a stability point of view. The learning approach is proposed to improve the performance by learning the error made between the expected trajectory and the obtained or by learn the payload dynamic in case of the payload transportation. It learns internal errors: simplifications, software offsets, unknown payload attached to the device and external disturbance such as ground effect or wind gusts.

V.8 Perspectives

The developed algorithms have been iteratively designed to improve the previous approaches by trying to correct the main weaknesses of the previous ones. However, the studies and experiments carried out have highlighted perspectives of work and avenues for improvements.

To start, we focus on a common point to all the presented algorithms. All the considered approaches inevitably require to learn a correction of an initial controller, in order to next being improved. This presents advantages on initial tuning simplicity. However, this has two disadvantages: the first is that we are dependent on the choice of this initial controller to flight and learn. It is therefore necessary to have a controller that works well enough in a lot of situations including difficult situations: for instance in a windy environment or under additional payload disturbances. The second drawback is a consequence of the previous one, which makes the proposed neural network dependent on the initial controller.

The first perspective could be to study the combination of the DNN with other types of initial controllers. For example, combining it with MPC. The use of the neural network could help to improve the prediction on the chosen horizon while reducing the computations. An alternative perspective considered, but not studied here, is a learning of the dynamics of the quadrotor in an independent way of the flight controller. For example, learning the functions f and g which constitute the quadrotor simplified dynamics $\dot{x} = f(x) + g(x)u$.

Another interesting research perspective is the deployment of outdoor flight algorithms. Indeed, the flight conditions are quite different in outdoor situations and an adaptation of the controller has to be expected. The GPS data is less accurate than that obtained from the motion capture room. More robust algorithms to estimate the position must be studied to perform these missions. In addition, the communication is probably much more disrupted in outdoor environment. It will require a more thorough study of possible delays and packet losses, in order to keep an efficient tracking.

An improvement of the approach would be to have a deep neural network that adapts to the quadrotor used. Indeed, the algorithms have been tested on a specific model of quadrotor. Thus, we can not use a network already learned into another quadrotor controller without relearning or re-adaptation beforehand. One way to consider generalization would be for example to increase the number of inputs of the neural network. It could take as parameter the ID of the considered device. However, this method would probably require a lot of flight data and a larger neural network to learn the differences between each model.

Some work can also be done using Reinforcement Learning algorithms. This approach constitutes a very interesting research axis. Indeed, finding RL solutions that can safely learn a controller without spending to much computation and parameter tuning is a very challenging task.

Finally, some work done during my thesis, but not presented in this manuscript, focused on the piloting of swarm of quadrotors. The work presented here could be applied and adapted to this particular case where disturbances between each drone are added.

Appendix A

Disturbances observers

A.1 Linear disturbances observer

Let's consider the system described by (A.1):

$$\begin{cases} \dot{x}(t) &= Ax(t) + Bu(t) + Dd(t), \\ y &= C(t)x(t), \end{cases} \tag{A.1}$$

with state $x(t) \in \mathbf{R}^n$, command input $u(t) \in \mathbf{R}^m$, measured signal $y(t) \in \mathbf{R}^p$ and disturbances $d(t) \in \mathbf{R}^q$ $A \in \mathbf{M}^n$, $B \in \mathbf{M}^{(m\times n)}$, $C \in \mathbf{M}^{(n\times p)}$, $D \in \mathbf{M}^{(q\times n)}$

Let's assume a model for disturbance d:

$$\dot{d}(t) = A_d d(t),$$

Defining an extended state as:

$$X(t) = \begin{bmatrix} x(t), & d(t) \end{bmatrix}^T.$$

One has the following extended system:

$$\begin{cases} \dot{X}(t) &= \begin{bmatrix} A & D \\ 0 & A_d \end{bmatrix} X(t) + \begin{bmatrix} B \\ 0 \end{bmatrix} u(t), \\ y(t) &= \begin{bmatrix} C & 0 \end{bmatrix} X(t). \end{cases} \tag{A.2}$$

By making the assumption that the system given in (A.2) is observable. The extended state observer is given by (A.3):

$$\dot{\hat{X}}(t) = A_e X(t) + B_e u(t) + L_{obs}(y(t) - C_e \hat{X}(t)), \tag{A.3}$$

where:

$$A_e = \begin{bmatrix} A & D \\ 0 & A_d \end{bmatrix}, \; B_e = \begin{bmatrix} B \\ 0 \end{bmatrix}, \text{ and } C_e = \begin{bmatrix} C & 0 \end{bmatrix}.$$

By denoting the observer error by $e(t) = X(t) - \hat{X}(t)$, the dynamic of observation error is

given by (A.4):

$$\dot{e}(t) = (A_e - L_{obs}C_e)e(t). \tag{A.4}$$

Then one can correctly chose the observer gain L_{obs} such that the observer error exponentially converges to 0.

Then one can get an estimation of d, that can be used as a feedforward term to cancel or at least minimize $\hat{d} - d$.

A.2 High-gain observer

In this appendix, we present high gain observer generalities. Equations and results mainly come from [Kha08]. Here we adapted the system model to fit our specific case Let's consider the nonlinear system given by (A.5):

$$\left\{ \begin{array}{rcl} \dot{x}_1(t) & = & x_2(t), \\ \dot{x}_2(t) & = & g(x(t), u(t)) + d(t), \end{array} \right. \tag{A.5}$$

Defining an extended state $x_3(t) = d(t)$ including disturbances, one has the following extended system (A.6):

$$\left\{ \begin{array}{rcl} \dot{x}_1(t) & = & x_2(t), \\ \dot{x}_2(t) & = & g(x(t), u(t)) + x_3(t), \\ \dot{x}_3(t) & = & \xi(t). \end{array} \right. \tag{A.6}$$

The function ξ is unknown, however thanks to the robustness of high-gain observers, one can estimate the extended state including disturbances. The extended high-gain observer can be taken as (A.7):

$$\left\{ \begin{array}{rcl} \dot{\hat{x}}_1(t) & = & \hat{x}_2(t) + \dfrac{\kappa_1}{\epsilon}(x_1(t) - \hat{x}_1), \\ \dot{\hat{x}}_2(t) & = & g(\hat{x}(t), u(t)) + \hat{x}_3(t) + \dfrac{\kappa_2}{\epsilon^2}(x_1(t) - \hat{x}_1), \\ \dot{\hat{x}}_3(t) & = & \dfrac{\kappa_3}{\epsilon^3}(x_1(t) - \hat{x}_1). \end{array} \right. \tag{A.7}$$

where κ_1, κ_2 and κ_3 are taken such as: the polynomial $p^3 + \kappa_1 p^2 + \kappa_2 p + \kappa_3$ is Hurwitz.

Appendix B

Deep Learning generalities

B.1 Introduction

This appendix focus on giving some generalities on Deep Learning and more particularly on deep feed-forward networks also known as multilayer perceptrons (MLPs).

Deep Learning is a branch of machine learning that uses artificial neural networks. It gathers different methods to model data combining complex architectures and nonlinear functions. The basis structure is the artificial neural network. Combining together several artificial neurons gives a deep neural network. It is directly inspired by human brain neurons, mimicking inter-neuronal communication. Deep neural network are composed of several layers interconnected in different ways, each layer receive and process data from previous to next layer. Deep Learning combined with large data allowed major advances in image, video and sound processing such as pattern recognition, image and text classification or natural language processing. These fields of application have since been widely developed in other fields including medicine, finance or robotics.

B.2 Deep neural networks

The aim of a feed forward neural network is to approximate some function f^*, it can be used either for classification and regression. It defines a nonlinear mapping between an input x to an output $y = f(x, \vartheta)$ and learns ϑ, the neural network parameters, to approximate the function f^*. To learn ϑ parameters, an optimization is performed using a training database $(\boldsymbol{x}_{\text{train}}, \boldsymbol{y}_{\text{train}})$. A deep neural network is composed of several layers basically built up with successive artificial neurons: the perceptrons.

B.2.1 Perceptrons description

The perceptron is the simplest artificial neural network at the basis of deep learning. It is a linear classifier, a function f of the input $x = [x_1, ..., x_n]^T$, weighted by $\boldsymbol{w} = (w_{ij})_{1 \leq m, 1 \leq n}$, with a bias $b = [b_1, ..., b_m]^T$. It is schematically represented as in Fig. B.1 and described by equation (B.1):

$$y = f(x) = \sigma(\boldsymbol{w}x + b) \tag{B.1}$$

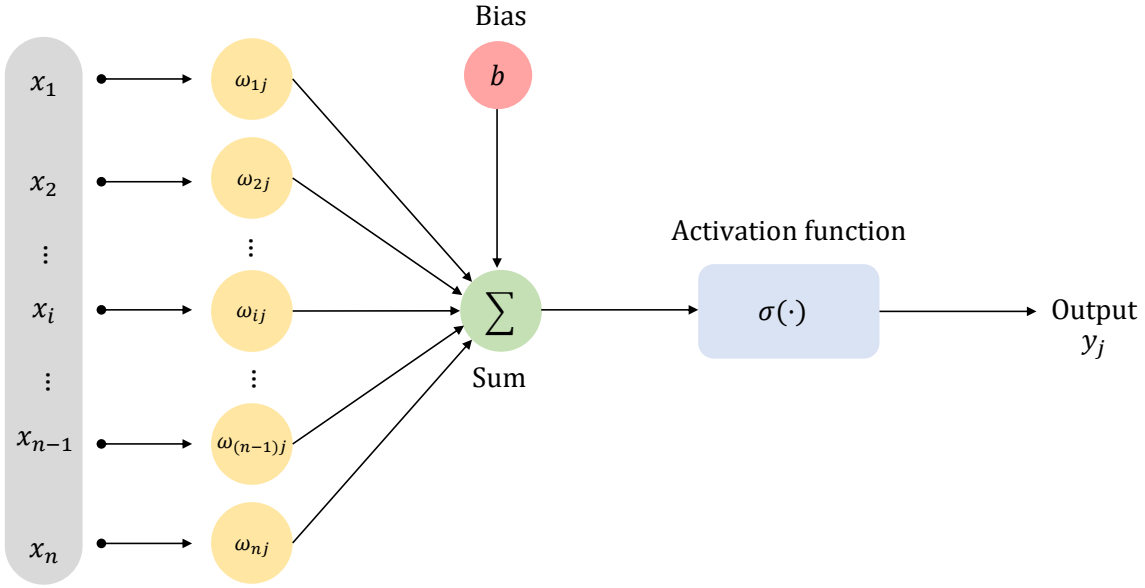


Figure B.1: Illustration of an artificial neural network

The nonlinear function σ is called *activation function*. It exists a lot of different activation functions in the deep learning literature. One can list few of them:

- Sigmoid: $\sigma(x) = \dfrac{1}{1 + exp(-x)}$,
- Hyperbolic tangent (tanh): $\sigma(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$,
- Rectified linear unit (ReLU): $\sigma(x) = \max(0, x)$,
- Leaky rectified linear unit (Leaky ReLU): $\sigma(x) = \begin{cases} 0.01x & \text{if } x < 0, \\ x & \text{if } x \geq 0, \end{cases}$
- Parametric rectified linear unit (Parametric ReLU): $\sigma(x) = \begin{cases} \alpha x & \text{if } x < 0, \\ x & \text{if } x \geq 0, \end{cases}$
- Gaussian Error Linear Unit (GELU): $\sigma(x) = \dfrac{1}{2}x(1 + erf(\dfrac{x}{\sqrt{2}}))$.

The presented activation functions are illustrated in Fig. B.2.

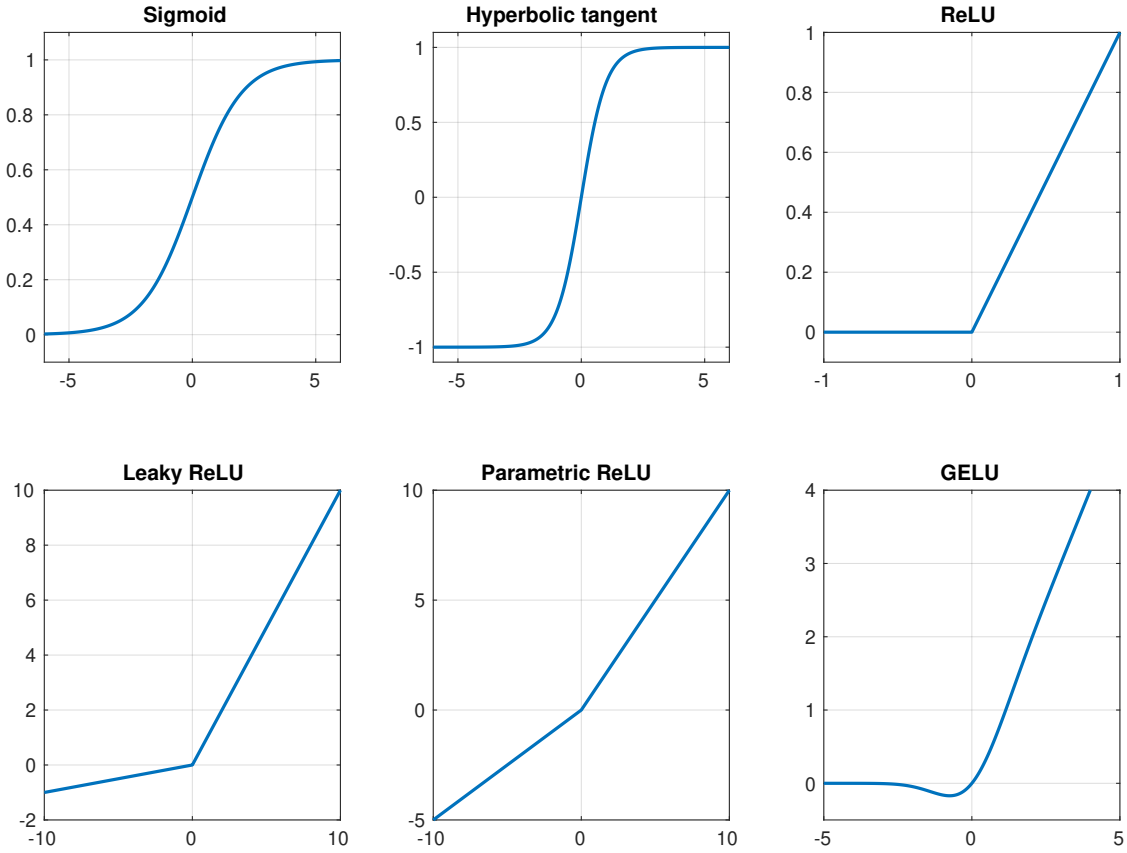


Figure B.2: Some activation functions for neural networks.

B.2.2 Multilayer perceptrons

A multilayer perceptron is built with a succession of layers. It can be represented as in eq. (B.2):

$$f(x) = f^{N_l}(f^{N_l-1}(...f^1(x))) \tag{B.2}$$

where N_l is the number of layers of the network. It means the number of nonlinear mappings. The first layer is called *input layer* and the last layer is called the *output layer*, here f^{N_l}. All other intermediate layers are called *hidden layers* (HL). An example of deep neural network structure is given in Fig. B.3.

It exists a lot of various neural network architectures. Here, we just provide some general insight of Deep Learning, and we will not go further into explaining them. It is worth mentioning: the convolutional neural networks, the recurrent neural networks, the generative adversarial networks, etc.

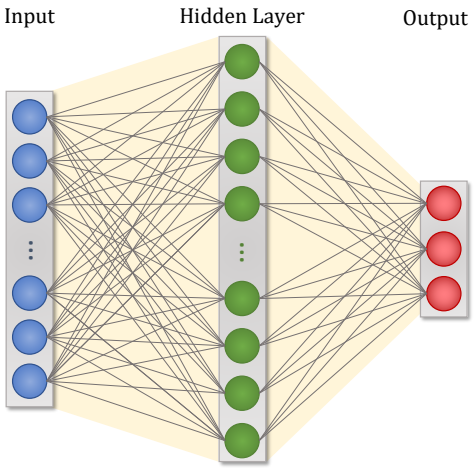


Figure B.3: An example of basic deep neural network.

B.3 Learning neural network parameters

The choice of the neural network constitute the first step in learning process. Next, all weights and biases of each layers, gathered in ϑ parameter, need to be estimated. To do so, an optimization problem is basically set, using a cost function called the *loss function*.

B.3.1 Loss function

The loss function is the quantity that the optimizer tends to minimize via ϑ parameter. It is very common to use the maximum likelihood estimation to find the best statistical parameters from a training dataset. The expected loss function is then given by:

$$\mathcal{L}(\vartheta) = \mathbf{E}_{(X,Y)\sim P}(\log(p_\vartheta(Y|X)). \tag{B.3}$$

For Gaussian model, minimizing eq. (B.3) is equivalent to minimize a quadratic loss:

$$\mathcal{L}(\vartheta) = \mathbf{E}_{(X,Y)\sim P}(\|Y - f(X,\vartheta)\|^2). \tag{B.4}$$

One can then derive an estimator of (B.4): the mean squared error. It is computing the average of the squared difference between the estimated and actual values:

$$\mathcal{L}(\vartheta) = MSE(\vartheta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \tilde{y})^2, \tag{B.5}$$

where $\tilde{y} = f(X,\vartheta)$ is the predicted value.

As for activation functions, it exists plenty different loss functions according the problem we are trying to deal with: regression, binary or multi-class classification.

B.3.2 Gradient descent algorithm

In order to minimize the loss function given by B.5, for multilayers perceptrons, gradient methods are used. The most popular way to perform it, is to use the stochastic gradient descent algorithm Stochastic Gradient Descent (SGD). Its batch version is presented in algorithm 1.

Algorithm 1 Stochastic Gradient Descent

Require: Set hyper-parameters: η
  Initialize $\vartheta = \vartheta_0$
  while Convergence criteria (epoch, gradient, etc.) **do**
    $\vartheta_{i+1} \leftarrow \vartheta_i - \eta \frac{1}{\#B} \sum_{j \in B} \nabla_\vartheta \mathcal{L}(f(X_j, \vartheta_i), Y_j)$
  end while

In this algorithm, B is a subset of the dataset $(\boldsymbol{x}_{\text{train}}, \boldsymbol{y}_{\text{train}})$, called a *batch*. An iteration over the whole dataset is called an *epoch*, it is a parameter to be fixed at the beginning of optimization.

Improved optimizers haven been explored, based on SGD. One can cite Nesterov-Accelerated Adaptive Moment Algorithm, refereed as NADAM, [Doz16]. This algorithm uses a momentum to move to an optimum, achieving faster convergence than SGD.

Algorithm 2 Nesterov-Accelerated Adaptive Moment

Require: Set hyper-parameters: μ, ν and l_r
  Initialize $\vartheta = \vartheta_0$
  Initialize $\mathrm{m} = \mathrm{m}_0$ and $\mathrm{v} = \mathrm{v}_0$
  while ϑ_i **do**
    $\mathrm{g}_i \leftarrow \nabla_\vartheta \mathcal{L}(\vartheta_{i-1})$
    Compute μ_k for $k = i$ and $k = i + 1$
      $\mu_k = \mu(1 - 0.5 \cdot 0.96^{k/250})$
    $\mathrm{m}_i \leftarrow \mu m_{i-1} + (1 - \mu) g_i$ ▷ Update first momentum
    $\mathrm{v}_i \leftarrow \nu v_{i-1} + (1 - \nu) g_i^2$ ▷ Update second momentum
    $\hat{\mathrm{m}}_\mathrm{i} \leftarrow \mu_{i+1} m_i / (1 - \prod_{j=1}^{i+1} \mu_j) + (1 - \mu_i) g_i / (1 - \prod_{j=1}^{i} \mu_j)$
    $\hat{\mathrm{v}}_\mathrm{i} \leftarrow \dfrac{\mathrm{v}_i}{1 - \nu^t}$
    $\vartheta_i \leftarrow \vartheta_{i-1} - l_r \dfrac{\hat{\mathrm{m}}_\mathrm{i}}{\sqrt{\hat{\mathrm{v}}_i} + \epsilon}$ ▷ Update optimization variable
  end while

More optimizers can be found in the following review [Rud16].

B.3.3 Computing gradient

As can be seen in the algorithms 1 and in algorithm 2, the knowledge of the gradient of the loss function with respect to ϑ parameter is needed: $\nabla_{\vartheta_i}\mathcal{L}(\vartheta_i)$. To do so, the *back-propagation* algorithm is used. Actually, it is a two phases process:

1. First a *forward pass* is performed to compute the output of the network and all intermediate values of each layers for the given ϑ.
2. Second a *backward pass* is using all the previously stored values to compute the intermediate values necessary to compute gradient of loss function.

B.4 Universal approximation theorem

The universal approximation theorem states that any continuous function on a compact of $\mathbf{R}^n$ can be approximated with a single layer network with a finite number of neurons. This result was first derived by Cybenko, for sigmoid activation.

Notations: Let $I_n = [0,1]^n$ denotes the n-dimensional unit cube $\mathbf{C}(I_n)$ denotes the space of continuous functions on I_n. $\mathbf{B}$ space of finite, signed regular measures on I_n

Definition B.1
σ is said discriminatory if for a measure $\mu \in \mathbf{B}(I_n)$,:

$$\int_{I_n} \sigma(a^T x + b)\mathrm{d}\mu(x) = 0$$

for all $y \in \mathbf{R}_n$, $b \in \mathbf{R}$ implies $\mu = 0$

Theorem B.1 (From Cybenko, [Cyb89])
Let σ be a continuous discriminatory function. The finite sum F of the following form:

$$F(x) = \sum_{i=1}^{N} \alpha_i \sigma(\omega_i^T x + \beta_i),$$

where $\omega_i \in \mathbf{R}^n$, $\alpha_i \in \mathbf{R}$, $\beta_i \in \mathbf{R}$ are dense in $\mathbf{C}(I_n)$.

Other formulation: Given any $f \in \mathbf{C}(I_n)$ and $\epsilon > 0$, there is a sum $F(x)$, of the above form, for which:

$$|F(x) - f(x)| < \epsilon \quad \text{for all } x \in I_n$$

It was then showed by Hornik, it is not specific to sigmoid. For multi-layers network, it was later proved by Zhou Lu et al. [Lu+17].

Bibliography

[Aba+15] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015 (cit. on p. 25).

[ACC21] Jorge M. Arizaga, Herman Castaneda, and Pedro Castillo. "Payload Swing Attenuation of a Fully-Actuated Hexacopter via Extended High Gain Observer Based Adaptive Sliding Control." In: *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, June 2021 (cit. on pp. 90, 105).

[ACL05] Kiam Heong Ang, G. Chong, and Yun Li. "PID control system analysis, design, and technology." In: *IEEE Transactions on Control Systems Technology* 13.4 (July 2005), pp. 559–576 (cit. on p. 45).

[ACN10] Pieter Abbeel, Adam Coates, and Andrew Y. Ng. "Autonomous Helicopter Aerobatics through Apprenticeship Learning." In: *The International Journal of Robotics Research* 29.13 (June 2010), pp. 1608–1639 (cit. on p. 45).

[ÅM21] Karl Johan Åström and Richard M Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2021 (cit. on p. 35).

[Ard18] Ardupilot. "ArduPilot Open Source Autopilot." In: (2018) (cit. on p. 45).

[Ban+16] Somil Bansal et al. "Learning quadrotor dynamics using neural network for flight control." In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, Dec. 2016 (cit. on pp. 45, 54).

[BCHB10] Morten Bisgaard, Anders la Cour-Harbo, and Jan Dimon Bendtsen. "Adaptive control system for autonomous helicopter slung load operations." In: *Control Engineering Practice* 18.7 (July 2010), pp. 800–811 (cit. on p. 90).

[BD18] Dario Brescianini and Raffaello D'Andrea. "Computationally Efficient Trajectory Generation for Fully Actuated Multirotor Vehicles." In: *IEEE Transactions on Robotics* 34.3 (June 2018), pp. 555–571 (cit. on p. 44).

[BHD13] Brescianini, Dario, Hehn, Markus, and D'Andrea, Raffaello. *Nonlinear Quadrocopter Attitude Control: Technical Report*. en. Tech. rep. 2013 (cit. on p. 11).

[Bir+11] Andreas Birk et al. "Safety, Security, and Rescue Missions with an Unmanned Aerial Vehicle (UAV)." In: *Journal of Intelligent & Robotic Systems* 64.1 (Jan. 2011), pp. 57–76 (cit. on p. 1).

[BK09] M. Bernard and K. Kondak. "Generic slung load transportation system using small size helicopters." In: *2009 IEEE International Conference on Robotics and Automation*. IEEE, May 2009 (cit. on p. 90).

[BM03] Andrew G. Barto and Sridhar Mahadevan. In: *Discrete Event Dynamic Systems* 13.4 (2003), pp. 341–379 (cit. on p. 70).

[BM12] Moses Bangura and Robert Mahony. "Nonlinear dynamic modeling for high performance control of a quadrotor." In: *Proc. of the Australasian Conference on Robotics and Automation*. Dec. 2012 (cit. on p. 44).

[BM14] Moses Bangura and Robert Mahony. "Real-time Model Predictive Control for Quadrotors." In: *IFAC Proceedings Volumes* 47.3 (2014), pp. 11773–11780 (cit. on p. 1).

[BMS04] S. Bouabdallah, P. Murrieri, and R. Siegwart. "Design and control of an indoor micro quadrotor." In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004.* IEEE, 2004 (cit. on p. 22).

[Bou07] Samir Bouabdallah. *Design and control of quadrotors with application to autonomous flying.* Tech. rep. Epfl, 2007 (cit. on p. 14).

[BS] S. Bouabdallah and R. Siegwart. "Backstepping and Sliding-mode Techniques Applied to an Indoor Micro Quadrotor." In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation.* IEEE (cit. on p. 1).

[Che+55] I. C. Cheeseman et al. *The effect of ground on a helicopter rotor in forward flight.* Tech. rep. 1955 (cit. on p. 16).

[Cho+18] Yinlam Chow et al. *A Lyapunov-based Approach to Safe Reinforcement Learning.* 2018 (cit. on p. 70).

[Cho+20] Jason Choi et al. *Reinforcement Learning for Safety-Critical Control under Model Uncertainty, using Control Lyapunov Functions and Control Barrier Functions.* 2020 (cit. on p. 70).

[Cyb89] G. Cybenko. "Approximation by superpositions of a sigmoidal function." In: *Mathematics of Control, Signals, and Systems* 2.4 (Dec. 1989), pp. 303–314 (cit. on p. 122).

[DAL13] Zachary T. Dydek, Anuradha M. Annaswamy, and Eugene Lavretsky. "Adaptive Control of Quadrotor UAVs: A Design Trade Study With Flight Evaluations." In: *IEEE Transactions on Control Systems Technology* 21.4 (July 2013), pp. 1400–1406 (cit. on p. 1).

[Dan+15] Li Danjun et al. "Autonomous landing of quadrotor based on ground effect modelling." In: *2015 34th Chinese Control Conference (CCC).* IEEE, July 2015 (cit. on p. 16).

[DLB14] Shicong Dai, Taeyoung Lee, and Dennis S. Bernstein. "Adaptive control of a quadrotor UAV transporting a cable-suspended load with unknown mass." In: *53rd IEEE Conference on Decision and Control.* IEEE, Dec. 2014 (cit. on pp. 90, 91).

[DO16] Jérôme Darbon and Stanley Osher. *Algorithms for Overcoming the Curse of Dimensionality for Certain Hamilton-Jacobi Equations Arising in Control Theory and Elsewhere.* 2016 (cit. on p. 70).

[Doz16] Timothy Dozat. "Incorporating nesterov momentum into adam." In: (2016) (cit. on p. 121).

[Dur+18] Sylvain Durand et al. "Event-Based PID Control: Application to a Mini Quadrotor Helicopter." In: *Journal of Control Engineering and Applied Informatics* 20.1 (Mar. 2018), pp. 36–47 (cit. on p. 1).

[Efe11] Mehmet Önder Efe. "Neural Network Assisted Computationally Simple PI$^\lambda$D$^\mu$ Control of a Quadrotor UAV." In: *IEEE Transactions on Industrial Informatics* 7.2 (May 2011), pp. 354–361 (cit. on pp. 2, 44).

[Fau+13] Aleksandra Faust et al. "Learning swing-free trajectories for UAVs with a suspended load." In: *2013 IEEE International Conference on Robotics and Automation*. IEEE, May 2013 (cit. on p. 91).

[Fau+17] Aleksandra Faust et al. "Automated aerial suspended cargo delivery through reinforcement learning." In: *Artificial Intelligence* 247 (June 2017), pp. 381–398 (cit. on p. 91).

[FFS18] Matthias Faessler, Antonio Franchi, and Davide Scaramuzza. "Differential Flatness of Quadrotor Dynamics Subject to Rotor Drag for Accurate Tracking of High-Speed Trajectories." In: *IEEE Robotics and Automation Letters* 3.2 (Apr. 2018), pp. 620–626 (cit. on p. 44).

[Fis+17] Jaime F. Fisac et al. *A General Safety Framework for Learning-Based Control in Uncertain Robotic Systems*. 2017 (cit. on p. 70).

[Foe+17] Philipp Foehn et al. "Fast Trajectory Optimization for Agile Quadrotor Maneuvers with a Cable-Suspended Payload." In: *Robotics: Science and Systems XIII*. Robotics: Science and Systems Foundation, July 2017 (cit. on p. 90).

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016 (cit. on p. 53).

[GC+11] J.F. Guerrero-Castellanos et al. "Bounded attitude control of rigid bodies: Real-time experimentation to a quadrotor mini-helicopter." In: *Control Engineering Practice* 19.8 (Aug. 2011), pp. 790–797 (cit. on p. 1).

[Gle+22] James Gleeson et al. *Optimizing Data Collection in Deep Reinforcement Learning*. 2022 (cit. on p. 70).

[GPS02] Herbert Goldstein, Charles Poole, and John Safko. *Classical mechanics*. American Association of Physics Teachers, 2002, p. 154 (cit. on p. 10).

[GT12] Jeremy H. Gillula and Claire J. Tomlin. "Guaranteed Safe Online Learning via Reachability: tracking a ground target using a quadrotor." In: *2012 IEEE International Conference on Robotics and Automation*. IEEE, May 2012 (cit. on p. 70).

[Gu+20] Weibin Gu et al. "Autonomous Wind Turbine Inspection using a Quadrotor." In: *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, Sept. 2020 (cit. on p. 1).

[Gue+15] M. E. Guerrero et al. "Passivity based control for a quadrotor UAV transporting a cable-suspended payload with minimum swing." In: *2015 54th IEEE Conference on Decision and Control (CDC)*. IEEE, Dec. 2015 (cit. on p. 90).

[Guo+20] Kexin Guo et al. "Multiple observers based anti-disturbance control for a quadrotor UAV against payload and wind disturbances." In: *Control Engineering Practice* 102 (Sept. 2020), p. 104560 (cit. on p. 90).

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." In: *Neural Networks* 2.5 (Jan. 1989), pp. 359–366 (cit. on p. 53).

[Hua+22] Hean Hua et al. "A Time-Optimal Trajectory Planning Strategy for an Aircraft With a Suspended Payload via Optimization and Learning Approaches." In: *IEEE Transactions on Control Systems Technology* 30.6 (Nov. 2022), pp. 2333–2343 (cit. on p. 91).

[Hun+92] K.J. Hunt et al. "Neural networks for control systems—A survey." In: *Automatica* 28.6 (Nov. 1992), pp. 1083–1112 (cit. on p. 44).

[Hwa+17] Jemin Hwangbo et al. "Control of a Quadrotor With Reinforcement Learning." In: *IEEE Robotics and Automation Letters* 2.4 (Oct. 2017), pp. 2096–2103 (cit. on pp. 2, 45).

[Kau+19] Elia Kaufmann et al. "Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing." In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, May 2019 (cit. on pp. 2, 44, 45).

[KCK13] Suseong Kim, Seungwon Choi, and H. Jin Kim. "Aerial manipulation using a quadrotor with a two DOF robotic arm." In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Nov. 2013 (cit. on p. 1).

[KH04] N. Koenig and A. Howard. "Design and use paradigms for gazebo, an open-source multi-robot simulator." In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. IEEE, 2004 (cit. on p. 28).

[Kha08] Hassan K. Khalil. "High-gain observers in nonlinear feedback control." In: *2008 International Conference on Control, Automation and Systems*. IEEE, Oct. 2008 (cit. on p. 116).

[Kol+19] Torsten Koller et al. *Learning-based Model Predictive Control for Safe Exploration and Reinforcement Learning*. 2019 (cit. on p. 70).

[Lam+19] Nathan O. Lambert et al. "Low-Level Control of a Quadrotor With Deep Model-Based Reinforcement Learning." In: *IEEE Robotics and Automation Letters* 4.4 (Oct. 2019), pp. 4224–4230 (cit. on pp. 2, 45).

[Lei06] Gordon J Leishman. *Principles of helicopter aerodynamics with CD extra*. Cambridge university press, 2006 (cit. on p. 13).

[Li+16] Qiyang Li et al. "Deep Neural Networks for Improved, Impromptu Trajectory Tracking of Quadrotors." In: *IEEE Int. Conf. on Robotics and Automation (ICRA)*. 2016. arXiv: `1610.06283 [cs.RO]` (cit. on p. 44).

[Lu+17] Zhou Lu et al. *The Expressive Power of Neural Networks: A View from the Width*. 2017 (cit. on p. 122).

[LZH21] Xiaoxuan Li, Jianlei Zhang, and Jianda Han. "Trajectory planning of load transportation with multi-quadrotors based on reinforcement learning algorithm." In: *Aerospace Science and Technology* 116 (Sept. 2021), p. 106887 (cit. on p. 91).

[Man15] Augustin Manecy. "Stratégies de guidage visuel bio-inspirées : application à la stabilisation d'un micro-drone et à la poursuite de cibles. (Bio-inspired visual strategies: application to stabilization of a micro UAV and to traget tracking)." PhD thesis. University of Grenobles, France, 2015 (cit. on pp. 14, 15, 17, 22).

[MCO19] Luís Martins, Carlos Cardeira, and Paulo Oliveira. "Linear Quadratic Regulator for Trajectory Tracking of a Quadrotor." In: *IFAC-PapersOnLine* 52.12 (2019), pp. 176–181 (cit. on p. 44).

[MHP15] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms." In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2015 (cit. on p. 25).

[MKC12] Robert Mahony, Vijay Kumar, and Peter Corke. "Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor." In: *IEEE Robotics & Automation Magazine* 19.3 (Sept. 2012), pp. 20–32 (cit. on pp. 14, 22).

[MW15] Nima Mohajerin and Steven L. Waslander. "Modelling a Quadrotor Vehicle Using a Modular Deep Recurrent Neural Network." In: *2015 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, Oct. 2015 (cit. on p. 45).

[Noa] *PX4/PX4-Autopilot software*. May 4, 2021 (cit. on pp. 25, 45).

[PA15] Ali Punjani and Pieter Abbeel. "Deep learning helicopter dynamics models." In: *IEEE Int. Conf. on Robotics and Automation (ICRA)*. 2015, pp. 3223–3230 (cit. on p. 45).

[PCF12] Ivana Palunko, Patricio Cruz, and Rafael Fierro. "Agile Load Transportation : Safe and Efficient Load Manipulation with Aerial Robots." In: *IEEE Robotics & Automation Magazine* 19.3 (Sept. 2012), pp. 69–79 (cit. on p. 90).

[PFC12] Ivana Palunko, Rafael Fierro, and Patricio Cruz. "Trajectory generation for swing-free maneuvers of a quadrotor with suspended payload: A dynamic programming approach." In: *2012 IEEE International Conference on Robotics and Automation*. IEEE, May 2012 (cit. on p. 90).

[Pi+20] Chen-Huan Pi et al. "Low-level autonomous control and tracking of quadrotor using reinforcement learning." In: *Control Engineering Practice* 95 (Feb. 2020), p. 104222 (cit. on p. 45).

[Pro95] Raymond W Prouty. *Helicopter performance, stability, and control*. 1995 (cit. on p. 17).

[Qui+09] Morgan Quigley et al. "ROS: an open-source Robot Operating System." In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5 (cit. on p. 28).

[RBR16] Charles Richter, Adam Bry, and Nicholas Roy. "Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments." In: *Springer Tracts in Advanced Robotics*. Springer International Publishing, 2016, pp. 649–666 (cit. on p. 44).

[Rob95] Anthony Robins. "Catastrophic Forgetting, Rehearsal and Pseudorehearsal." In: *Connection Science* 7.2 (June 1995), pp. 123–146 (cit. on p. 79).

[Rud16] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2016 (cit. on p. 121).

[Sal+10] Atheer L. Salih et al. "Modelling and PID controller design for a quadrotor unmanned air vehicle." In: *2010 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*. IEEE, May 2010 (cit. on p. 1).

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement Learning*. 2nd ed. Adaptive Computation and Machine Learning series. Cambridge, MA: Bradford Books, Nov. 2018 (cit. on p. 45).

[Sch20] David Schneider. "The delivery drones are coming." In: *IEEE Spectrum* 57.1 (Jan. 2020), pp. 28–29 (cit. on p. 1).

[SG64] Abraham. Savitzky and M. J. E. Golay. "Smoothing and Differentiation of Data by Simplified Least Squares Procedures." In: *Analytical Chemistry* 36.8 (July 1964), pp. 1627–1639 (cit. on p. 40).

[Sha+19] Hazim Shakhatreh et al. "Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges." In: *IEEE Access* 7 (2019), pp. 48572–48634 (cit. on p. 1).

[She20] Alex Sherstinsky. "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network." In: *Physica D: Nonlinear Phenomena* 404 (Mar. 2020), p. 132306 (cit. on p. 101).

[Shi+19] Guanya Shi et al. "Neural Lander: Stable Drone Landing Control Using Learned Dynamics." In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, May 2019 (cit. on pp. 2, 45, 54).

[Sil+21] Giuseppe Silano et al. "A Multi-Layer Software Architecture for Aerial Cognitive Multi-Robot Systems in Power Line Inspection Tasks." In: *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, June 2021 (cit. on p. 1).

[Str+06] Alexander L. Strehl et al. "PAC model-free reinforcement learning." In: *Proceedings of the 23rd international conference on Machine learning - ICML '06*. ACM Press, 2006 (cit. on p. 70).

[Tay+19] Andrew J. Taylor et al. "Episodic Learning with Control Lyapunov Functions for Uncertain Robotic Systems." In: (2019) (cit. on p. 70).

[TG+17] Arturo Torres-González et al. "A Multidrone Approach for Autonomous Cinematography Planning." In: *ROBOT 2017: Third Iberian Robotics Conference*. Springer International Publishing, Nov. 2017, pp. 337–349 (cit. on p. 1).

[Tor+21] Guillem Torrente et al. "Data-Driven MPC for Quadrotors." In: *IEEE Robotics and Automation Letters* (2021) (cit. on p. 44).

[UE20] J B Ubbink and J A A Engelbrecht. "Sequence-Constrained Trajectory Planning and Execution for a Quadrotor UAV with Suspended Payload." In: *IFAC-PapersOnLine* 53.2 (2020), pp. 9405–9411 (cit. on p. 90).

[Voo09] Holger Voos. "Nonlinear control of a quadrotor micro-UAV using feedback-linearization." In: *2009 IEEE International Conference on Mechatronics*. IEEE, 2009 (cit. on p. 96).

[Wan+15] Yaonan Wang et al. "Fuzzy radial basis function neural network PID control system for a quadrotor UAV based on particle swarm optimization." In: *2015 IEEE International Conference on Information and Automation*. IEEE, Aug. 2015 (cit. on pp. 2, 44).

[Wan+20] Chuanzheng Wang et al. *Learning Control Barrier Functions with High Relative Degree for Safety-Critical Control*. 2020 (cit. on p. 70).

[WZ21] Kim Peter Wabersich and Melanie N. Zeilinger. "A predictive safety filter for learning-based control of constrained nonlinear dynamical systems." In: *Automatica* 129 (July 2021), p. 109597 (cit. on p. 70).

[Yu+22] Gan Yu et al. "Aggressive maneuvers for a quadrotor-slung-load system through fast trajectory generation and tracking." In: *Autonomous Robots* 46.4 (Mar. 2022), pp. 499–513 (cit. on p. 90).

[Zen+20] Jun Zeng et al. "Differential Flatness Based Path Planning With Direct Collocation on Hybrid Modes for a Quadrotor With a Cable-Suspended Payload." In: *IEEE Robotics and Automation Letters* 5.2 (Apr. 2020), pp. 3074–3081 (cit. on p. 90).

[Zha+22] Dingqi Zhang et al. *A Zero-Shot Adaptive Quadcopter Controller*. 2022 (cit. on p. 2).

[Zho+16] Xu Zhou et al. "Stabilization of a Quadrotor With Uncertain Suspended Load Using Sliding Mode Control." In: *Volume 5A: 40th Mechanisms and Robotics Conference*. American Society of Mechanical Engineers, Aug. 2016 (cit. on p. 90).

[ZN93] J. G. Ziegler and N. B. Nichols. "Optimum Settings for Automatic Controllers." In: *Journal of Dynamic Systems, Measurement, and Control* 115.2B (June 1993), pp. 220–222 (cit. on p. 36).

[ZS19] Jun Zeng and Koushil Sreenath. "Geometric Control of a Quadrotor with a Load Suspended from an Offset." In: *2019 American Control Conference (ACC)*. IEEE, July 2019 (cit. on p. 90).

Résumé — Ces dernières années ont vu l'attrait des drones croître exponentiellement grâce à leur facilité de construction et leur grande efficacité en vol. Cette technologie, ainsi que ses applications, ne cessent d'évoluer depuis. La réalisation d'autopilotes efficaces est ainsi une source de recherche très active. En effet, le drone doit être capable de réagir à de nombreuses sources de perturbations possibles. Avec les succès récents de l'intelligence artificielle dans de nombreux domaines, la réalisation de tels algorithmes de vol prend une toute nouvelle dimension, permettant ainsi une meilleure intégration de ces perturbations.

L'objectif de cette thèse consiste en la réalisation d'algorithmes d'autopilote intelligent de drones, en particulier pour des quadrirotors. L'idée principale de ces travaux est de mixer les approches standards de contrôle et les méthodes d'apprentissages pour améliorer l'efficacité et la performance des vols de drone quadrirotors. Cette amélioration passe par l'apprentissage des différentes perturbations externes ou internes que le système subit en vol. Elles sont ensuite estimées et compensées dans la commande. Par ailleurs, l'objectif est de proposer des algorithmes d'apprentissage facilement implémentables sur des contrôleurs standard de drone.

La première solution proposée est basée sur le couplage d'un réseau de neurones profond et d'une architecture de PID en cascade. Le réseau de neurones a pour objectif d'apprendre les erreurs à un modèle linéaire du quadrirotor. Ce modèle linéaire est utilisé pour l'établissement des performances désirées. Ces erreurs contiennent les effets non pris en compte dans le contrôleur initial, les erreurs de linéarisation, mais également les erreurs liées aux perturbations externes. Une fois l'apprentissage fait et intégré au contrôleur de vol, le comportement du quadrirotor se rapproche du comportement initialement désiré. Cette approche est ensuite améliorée afin de proposer une solution en ligne, basée sur événements. Elle tente d'améliorer deux aspects de la précédente : les conditions de stabilité limitées et l'apprentissage hors ligne du réseau de neurones. Elle repose sur des critères de performance et de stabilité du système en boucle fermée. La solution proposée se base sur une succession de collecte et d'apprentissage en vol, pour améliorer l'efficacité du vol en continu. Enfin, le cas du transport de charge suspendue inconnue est abordée. Cette tâche génère des perturbations internes importantes complexe à maîtriser. Ainsi, un algorithme de linéarisation du système non-linéaire, assisté par un réseau de neurones fenêtré, est proposé pour réaliser cette mission. L'algorithme proposé permet un suivi de trajectoire amélioré et moins oscillants lors du transport de la charge suspendue.

Tous les contrôleurs proposés sont validés en simulation et testés expérimentalement dans la plateforme expérimentale de motion capture du GIPSA-lab.

Mots clés : quadrirotor, suivi de trajectoire, contrôle basé sur modèle, apprentissage profond, rejet de perturbations

Abstract — The last couple of years have seen the interest in drones grow exponentially due to their ease of construction and high flight efficiency. This technology, as well as its applications, have been evolving ever since. The development of efficient autopilots is thus a very active source of research. Indeed, the drone must be able to be reactive to many possible sources of disturbances. With the recent success of artificial intelligence in many domains, the development of such flight algorithms takes a whole new dimension, allowing a better handling of these disturbances.

The objective of this thesis is the development of intelligent autopilot algorithms for UAVs, in particular for quadrotors. The main idea of this work is to merge standard control approaches and learning methods to improve the efficiency and performance of quadrotor flights. This improvement is achieved by learning the different external or internal disturbances that the system experiences during flights. They are then estimated and compensated within the command. Moreover, the objective is to propose learning algorithms that can be easily implemented on standard controllers.

The first proposed solution is based on the coupling of a deep neural network and a cascade PID architecture. The neural network aims at learning errors to a linear model. This linear model is used for tuning the desired tracking performance. Learned errors include effects not taken into account in the initial controller, the linearization errors, but also errors related to external disturbances. Once the learning is done and integrated in the flight controller, the quadrotor better fits the initially desired behavior. This approach is then improved to propose an online event-based solution. This second solution tries to correct two aspects of the previous one: the limited stability insurance and the offline learning of the deep neural network. It is based on performance and stability criteria of the closed-loop system. The proposed solution relies on a succession of in-flight data collection and learning, in order to continuously improve the quadrotor behavior. Finally, the case of unknown suspended payload transport is addressed. This task generates significant internal disturbances that are complex to handle. Thus, a linearization algorithm of the non-linear system, assisted by a windowed deep neural network, is proposed to achieve it. The algorithm allows an improved and less oscillating trajectory tracking while transporting the suspended payload.

All the proposed controllers are validated in simulation and tested experimentally in the experimental motion capture platform of GIPSA-lab.

Keywords: quadrotor, trajectory tracking, model-based control, deep learning, disturbance rejection

GIPSA-Lab, 11 rue des Mathématiques
38400 Saint-Martin d'Hères, France