Neural Disturbance Rejection for a Multirotor

by

Henry Kotzé

Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Electronic) in the Faculty of Engineering at Stellenbosch University

Supervisor:	Dr H. W. Jordaan
Co-supervisor:	Dr H. Kamper

March 2021

Plagiaatverklaring / *Plagiarism Declaration*

1. Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.
   Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as my own.

2. Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit 'n vorm van diefstal is.
   I agree that plagiarism is a punishable offence because it constitutes theft.

3. Ek verstaan ook dat direkte vertalings plagiaat is.
   I also understand that direct translations are plagiarism.

4. Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelikse aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.
   Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism

5. Ek verklaar dat die werk in hierdie skryfstuk vervat, behalwe waar anders aangedui, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik ingehandig het vir bepunting in hierdie module/werkstuk of 'n ander module/werkstuk nie.
   I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

Studentenommer / *Student number*	Handtekening / *Signature*
Voorletters en van / *Initials and surname*	Datum / *Date*

Abstract

Neural Disturbance Rejection for a Multirotor

Henry Kotzé

Department of Electrical and Electronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Thesis: MEng (Electronic)

March 2021

The thesis addresses the problem of multirotors experiencing various disturbances such as wind, payloads and ground effects. These disturbances introduce challenges during specific application uses such as delivery, capturing images and line following. The project models these disturbances as unknown and attempts to implement a controller architecture which rejects them to provide a general solution for all application uses.

The project has a particular focus on using neural networks as a solution to the problem because of the recent advances the technique has made in fields which share common attributes. Existing approaches mostly attempt to replace the controller entirely with neural networks, because of its ability to learn nonlinear behaviour, which many classical controllers ignore. This project rather focuses on augmenting the classical controller with neural networks to account for disturbances and nonlinear behaviour. Specifically, the project uses a disturbance rejection architecture using a neural network as its observer for disturbances. The neural network estimates the disturbances which are then rejected by feeding it back into the classical controller output signal.

Synthetic labelled data is generated using the Gazebo simulation environment wherein disturbances of a specific nature occur with domain randomisation applied for Sim2Real transfer. The flight controllers used is PX4 which provides the Software-in-the-Loop functionality to fly a multirotor along a specific trajectory. The neural network estimation for practical flights shows good Sim2Real transfer with its ability to estimate payloads being carried by a multirotor and ground effects during landing. The neural network disturbance rejection is also compared to two other classical observers, namely the Extended Kalman Filter (EKF) and the Extended State Observer (ESO). The neural network shows superior disturbance rejection over the EKF and ESO when the multirotor is experiencing force disturbances. For torque disturbances, the ESO performed the best. From the disturbance rejection results, it is evident that for torque disturbances which influence the faster dynamics of the multirotor, observers should execute alongside the controllers such as the ESO. For disturbances which influence the slower dynamics of the multirotor, algorithms which execute on a companion board are sufficient and better. Specifically, the use of a neural network as an observer in a disturbance rejection architecture shows compelling evidence as the method for rejecting unknown disturbances influencing a multirotor.

Uittreksel

Steurseinverwerping vir 'n Multirotor Hommeltuig deur middel van Neural Netwerke

("Neural Disturbance Rejection for a Multirotor")

Henry Kotzé

Departement Elektroniese en Elektroniese Ingenieurswese,
Universiteit van Stellenbosch,
Privaatsak X1, Matieland 7602, Suid Afrika.

Tesis: MIng (Elektronies)

Maart 2021

Die tesis pak die probleem aan dat hommeltuie verskeie versteurings ondergaan tydens 'n vlug. Hierdie versteurings kan wind, grond effekte en vragte insluit wat problematies is vir wanneer hommeltuie in verskeie praktiese toepassings gebruik word. Die projek benader hierdie versteurings as onbekend en beplan om a beheer argitektuur te ontwikkel wat 'n algemene oplossing bied vir hommeltuie wat versteurings ervaar tydens praktiese vlugte.

Die projek fokus om neurale netwerke te gebruik as deel van die oplossing as gevolg van die onlangse vordering wat neurale netwerke gemaak het in velde wat dieselfde eienskappe as die van beheerstelsels het. Bestaande tegnieke benader die probleem deur om die klassieke beheerder heeltemal te vervang met neurale netwerke weens die voordele wat neurale netwerke bied vir nie-linieëre gedrag. Die projek benader die probleem deur die klassieke beheerder saam met 'n neurale netwerk te werk om die versteurings en nie-linieëre gedrag te beveg. Die projek gebruik 'n versteuring verwerping argitektuur wat 'n neurale netwerk gebruik as sy versteuring afskatter. Die neurale netwerk skat die versteurings af wat dan in die terugvoer lus gebruik word met die klassieke beheerder se uitree sein.

Die neurale netwerk word geleer deur gebruik te maak van die Gazebo simulasie omgewing om sintetiese data te genereer. Die simulasie omgewing word verder verryk deur om omgewings ewekansigheid toe te pas om sodoende die neurale netwerk se simulasie-tot-werklikheid skakel te verbeter. Die PX4 vlugbeheerder word gebruik om die hommeltuig in simulasie te laat vlieg. Die neurale netwerk se afskatting van versteurings op praktiese vlugtoetse wys dat die neurale network goed oorgeskakel het na die werklikheid deurdat dit ’n vrag wat deur die hommeltuig gedra word kan afskat asook grond effekte. Die neurale netwerk word ook vergelyk teen twee ander klassieke tegnieke: die Uitgebreide Kalman Filter (UKF) en die Uitgebreide Toestand Waarnemer (UTW). Die neurale netwerk se versteuring verwerping is beter as die van UKF asook die UTW wanneer die hommeltuig onderworpe is aan krag versteurings. Vir torsie versteurings is die UTW beste. Die versteuring verwerping resultate toon aan dat vir torsie versteuring is dit beter om waarnemers soos die UTW te gebruik wat op die vlugbeheer stelsel uitgevoer word. Stadige versteurings soos die van krag versteurings kan verwerp word deur gebruik te maak van algoritmes wat meer kragtige verwerkingseenheid stelsels kort. Spesifiek toon die resultate aan dat die gebruik van ’n neurale netwerk as ’n versteuring afskatter in ’n versteuring verwerping argitektuur die voorkeur geniet.

Acknowledgements

I would like to express my sincere gratitude to the following people and organisations:

- I lift up my eyes to the mountains - where does my help come from? My help comes from the Lord, the Maker of heaven and earth.
  - Psalm 121v1-2
- Dr Willem Jordaan & Dr Herman Kamper for the supervision during the two years.
- Dr Japie Engelbrecht for organising funding.
- Reghard Grobler, Armand Scholts, Ruan Viljoen, Johan Ubbink, Francois Slabber, Daniel Jansen, Martin Babl, Victor Sciocatti
- Anton Erasmus for allowing me to use his Tikz drawings.
- The academic staff for asking questions during research group meetings.
- My suster, Liesl who helped with proof reading.
- Family and friends.

Contents

List of Figures

List of Tables

Nomenclature

Constants

g =	$9.81\,\mathrm{m/s^2}$

Variables

b	Bias in a neural network	[ ]
m	Mass	[kg]
n	Batchsize	[ ]
p	Dropout probability	[ ]
s	Standard deviation	[ ]
t	Time	[s]
w	Weight in a neural network	[ ]
F	Force	[N]
M	Moment	[N·m]
N	Windowsize	[ ]
W	Mathematical operation of NN unit. See Equation 2.7	[ ]
Z	Placeholder for $\boldsymbol{X}$ or $\boldsymbol{q}$	[ ]

α	Learning rate	[ ]
β	Number of ReLU units	[ ]
γ	Number of LSTM units	[ ]
δ	Virtual output of PX4 controllers.	[ ]
θ	Rotation angle	[rad]
λ	Weight regularisation coefficient	[ ]
μ	Mean	[ ]
σ	Activation function	[ ]
ϕ	Rotation angle	[rad]
ψ	Rotation angle	[rad]
Θ	Neural network	[ ]
$\mathcal{N}$	Gaussian distribution	[ ]

Vectors

$\boldsymbol{c}$	The unit state of a RNN unit
$\boldsymbol{f}$	Output of forget gate of RNN unit
$\boldsymbol{i}$	Output of *tanh* layer of RNN unit
$\boldsymbol{j}$	Output of input gate layer of RNN unit
$\boldsymbol{o}$	The output state of a RNN unit
$\boldsymbol{q}$	Unit quaternion
$\boldsymbol{x}$	Input matrix for a neural network

$\boldsymbol{y}$	Output of neural network
$\boldsymbol{z}$	Input vector
$\boldsymbol{F}$	Discrete Jacobian matrix of nonlinear process model
$\boldsymbol{H}$	Discrete Jacobian matrix of nonlinear measurement model
$\boldsymbol{I}$	Moment of inertia matrix
$\boldsymbol{K}$	Normalising PX4 gain
$\boldsymbol{L}$	Estimator full state gain
$\boldsymbol{Q}$	Measurement noise matrix
$\boldsymbol{R}$	Process noise matrix
$\boldsymbol{V}$	Velocity vector
$\boldsymbol{X}$	Position vector
$\boldsymbol{\Omega}$	Angular rate vector
$\bar{\boldsymbol{x}}$	Unit position vector
$\bar{\boldsymbol{y}}$	Unit position vector
$\bar{\boldsymbol{z}}$	Unit position vector

Subscripts

c	The *tanh* layer of a RNN
f	Forget gate of a RNN
i	Placeholder for one of the unit position vectors
j	The unit in the l layer of a neural network

k	The unit in the l+1 layer of a neural network
o	Output gate of a RNN
r	Reference signal
x	Input gate of a RNN
$\mathcal{B}$	Body frame axis
$\mathcal{I}$	Inertial frame axis

Superscripts

l	The layer in a neural network
D	Disturbances
G	Gravity
T	Thrust
$+$	Current timestep
$-$	Previous timestep

Abbreviations

6DoF	Six Degree of Freedom
CoG	Center of Gravity
COTS	Commercial Off The Shelve
DCM	Direct Cosine Matrix
EKF	Extended Kalman Filter
ESO	Extended State Observer

GAP	Gazebo Awesome Plugins
GPS	Global Positioning System
GRU	Gated Recurrent Unit
HPC	High Performance Computer
IAE	Integrated Absolute Error
IMU	Inertial Measurement Unit
ITAE	Integrated Time Absolute Error
LSTM	Long Short Term Memory
LPF	Low Pass Filter
MAE	Mean Absolute Error
MSE	Mean Squared Error
NaN	Not a Number
NDI	Nonlinear Dynamic Inversion
NED	North-East-Down
NN	Neural Network
PID	Proportional Integrated Derivative
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
ROS	Robotic Operating System
RTOS	Real Time Operating System

RUAV	Rotary Wing Unmanned Vehicle
SISO	Single-Input-Single-Output
SITL	Software in the Loop
UKF	Unscented Kalman Filter

Chapter 1

Introduction

Aviation consists of many vehicles which are classified based on vehicle characteristics and operating airspace. One of these vehicles, formerly known as Rotary Wing Unmanned Aerial Vehicle (RUAV) is better known to the consumer market as drones. RUAVs can be described by their primary method of propulsion: rotating propellers and the use of differential thrust produced by these propellers to translate and orient the vehicle, as shown in Fig. (1.1). Collectively, drones are appropriately described as multirotors and can further be categorised based on the number of propellers they use, i.e. quadcopter for four propellers and octocopter for eight.

Figure 1.1: A quadcopter, which is a subset of multirotors, hovering in the air.

1.1 Motivation

A number of industries have started to incorporate multirotors into their workflow. Reasons why these industries are increasingly using this technology includes lower operation cost, faster deployment in comparison to the more traditional options, and improved decision making. The motivation and challenges for using multirotors in each application area are described below.

1.1.1 Agriculture

The agriculture sector has mainly introduced multirotors in the area of crop analysis. Various companies fly multirotors above the crops and with the use of special sensors can estimate crop growth and crop stress [3]. These estimates effectively lead to better decision making and accurate use of pesticides. Multirotors have been used to irrigate crops as shown in Fig. (1.2) which provides a significant improvement in the response time of irrigation and operation costs as opposed to the traditional alternative of fixed-wing aeroplanes.

Weather conditions pose a challenge for multirotors in these applications. Multirotors are sensitive to wind and this influences their flight time and accuracy, which is essential to completing its mission successfully. For multirotors to enter this market they must improve their ability to fly in unfavourable weather conditions.

1.1.2 Pipe and Gas Industry

Multirotors have the advantage of reaching areas which are difficult for humans to access. This is seen by multirotors inspecting large structures such as pipes

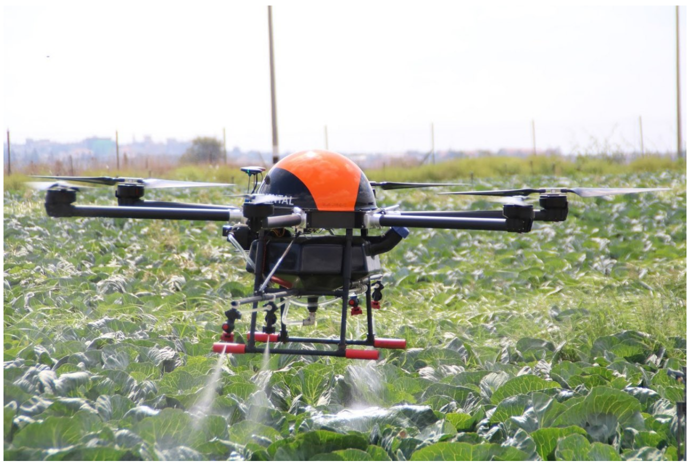


Figure 1.2: A multirotor irrigating crops.

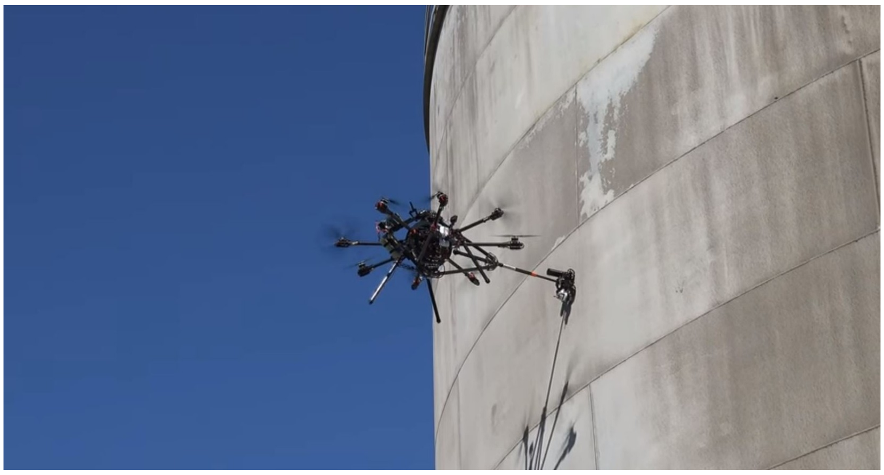


Figure 1.3: Multirotor inspecting a wall.

and walls, as shown in Fig. (1.3). The multirotor uses a camera to capture the area of interest, and then a human is able to inspect it from the safety of their office to determine whether repairs are needed. This greatly reduces operational costs and improves worker safety [4].

Flying near walls and objects presents a challenge for multirotors since the airflow produced by their propellers collide with the surface and flow back towards the multirotor, causing unsteady motion. This unsteady motion near surfaces makes sensor measurements and capturing images more difficult.

1.1.3 Search and Rescue

Multirotors are introduced into security and emergency services in which fast reaction time is critical to successfully prevent disasters [5]. The use of multirotors in disaster relief is shown in Fig. (1.4) where a multirotor is used to bring a life-saver to a human in distress.

For missions where human lives are at risk or too dangerous for humans to enter, results in fault-tolerant and all-weather systems. Multirotors must be able to absorb a motor failure and withstand severe weather conditions during times of emergencies to provide the reliability when humans lives are endangered.

1.1.4 Consumer Market

Delivering consumer goods with the use of multirotors has been a near-future prospect with the technology becoming more mature and reliable. Companies such as Amazon have recently attained acceptance from regulators, allowing them to operate multirotors autonomously [6]. Using multirotors in the deliv-

Figure 1.4: Multirotor being used in a sea rescue mission.

ery of packages removes the barrier of traffic caused by cars and provides the customer with an accurate time of delivery.

Delivering packages consists of flying with payloads which are suspended or directly attached to the body of the vehicle and which vary in size and weight. These varying parameters create challenges to the stability, performance and flight time of the multirotors, all of which are important for safety and economies of scale. Incremental improvements in these components lead to considerable improvements for company margins.

1.1.5 Challenges

The increasing demand for multirotors has resulted in development to application-specific challenges. These challenges include flying with a payload for delivering, flying near surfaces for inspection and flying in unfavourable weather to attain a 24/7 availability. All of these challenges can be described as disturbances influencing the multirotor as they do not form part of the general flight conditions in which these multirotors were initially designed. This points to the following question: If disturbances in all the forms which it arises could be rejected, would this provide feasible solutions to the various sectors?

1.2 Problem Definition

The project aims to design and implement a controller architecture for rejecting various unknown disturbances affecting a multirotor during a stable flight. These unknown disturbances will be in the form of forces and torques influencing the multirotor in all three body-axis directions.

1.3 Approach

The problem is solved by dividing the project into the following steps:

1. There is a particular focus on using neural networks as part of the proposed solution to the problem. The focus to use neural networks is driven by the fact that this technique has made recent advances in numerous fields [7]. These numerous fields share common attributes to the field of control systems and therefore provide confidence that improvement is probable. Thus, the use of neural networks in the control, state estimation and disturbance rejection is investigated on an inverted pendulum. The inverted pendulum was chosen as a testing ground for neural networks in a control system environment due to its intuitive dynamics. The results generated during this work are not presented due to it being out of the scope of the project, however it guided many decisions which were made during the project. These results were published in a conference paper at the **International Federation for Automatic Control (IFAC) World Congress 2020**. The conference paper was titled "Training neural networks for estimation, control and disturbance rejection" and is cited as Kotzé *et al.* [8].

2. Establish the current approaches of rejecting disturbances influencing a multirotor by reviewing the literature.

3. Formulate a proposed solution using the literature study of disturbance rejection for multirotors. The formulation of the proposed solution was biased towards containing neural networks as mentioned.

4. Research and implement the required techniques and overall system to demonstrate the proposed solution.

5. Compare the proposed solution to classical approaches.

1.4 Thesis Outline

Chapter 2 presents background on how disturbances arise in systems, which is followed by a literature study of the current approaches of disturbance rejection for multirotors. The chapter then presents the proposed solution and the supporting technical background. The proposed solution is to use a neural network in a disturbance rejection architecture.

Chapter 3 presents an overview of the system used to solve the problem. The system contains many components which operate together in a specific order to create the workflow of the project. These components fulfil specific roles to ensure that the proposed solution is successfully implemented.

Chapter 4 elaborates on key components mentioned in Chapter 3, which consist of features that are concerned with transferring a neural network trained in simulation to practical tests.

Chapter 5 presents the estimation of various disturbances by the neural network and by two other traditional solutions without integrating them into the selected control architecture. These other two solutions are the Extended State Observer (ESO) and the Extended Kalman Filter (EKF).

Chapter 6 presents the neural network, ESO, and EKF integrated into the proposed controller architecture. The neural network, ESO, and EKF are compared using a quantitative method to score their performance which enables commentary to be given.

Chapter 7 concludes the project with a summary of what was achieved during the project, a recommendation is given and commentary is provided for future work to improve and build on the results presented.

Chapter 2

Background

This chapter will introduce how disturbances originate from a control systems perspective and is followed by a literature study focusing on how control systems reject them. A tree diagram is included to summarise the various branches of disturbance rejection approaches. Following the literature study, which provides supporting evidence, the suggested solution to the problem definition is introduced. The chapter then presents the required technical concepts in order to understand how the solution will be implemented.

2.1 Origin of Disturbances

Disturbances originate during the modelling process when certain dynamics are unknown, omitted, and assumptions and simplifications are made to create a tractable problem. These omissions and assumptions result in a mathematical model which deviates from experimental results. These deviations from the mathematical model are an indication of disturbances and are categorised in the following manner.

Disturbances can be categorised as either external or internal, and this categorisation is dependent on the modelling of the system. The modelling of a multirotor involves assuming rigid body dynamics and no deformation in the structural members of the multirotor. For small multirotors this is valid, but as they increase in size, the vibration of the structural members becomes significant and more structural reinforcement is required. The vibration of structural members is a result of the control inputs exciting their natural frequencies. These type of disturbances are seen as internal because they are excited by the control system, which does not take these unmodelled dynamics into account. These disturbances are not modelled in a simulation environment and can only be identified during an experimental flight and are removed by the use of filters such as a band rejection filter.

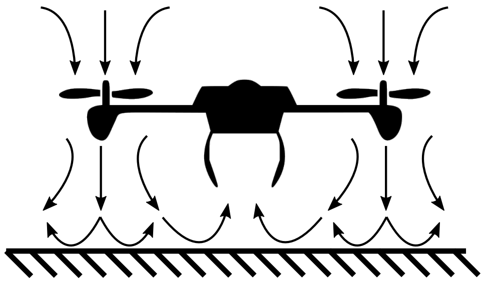


Figure 2.1: The airflow produced by the propeller is being washed up by the surface onto the multirotor. This phenomenon is known as ground effects.

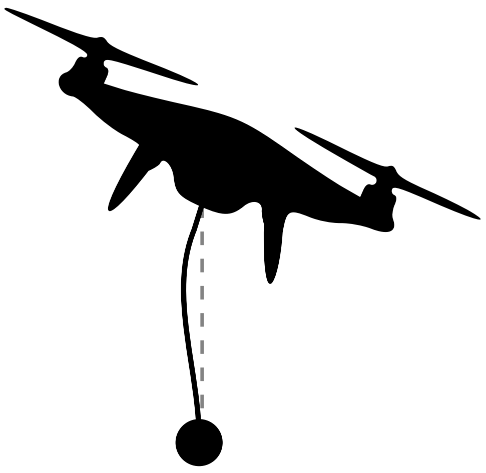


Figure 2.2: Multirotor carrying a suspended payload[1].

External disturbances are forces and moments affecting the multirotor from the environment it is operating in. The most common disturbance is the effect of wind which is a stochastic process omitted during modelling. Other external disturbances include ground effects which are upwash flow from the propellers near walls, as shown in Fig. (2.1) or suspended payloads attached to the multirotor shown in Fig. (2.2) which is omitted during modelling. The influence of external disturbances can be tested before experiments in simulation to provide limits on the disturbance rejection capabilities of the control system.

2.2 Literature Study

Rejecting disturbances which influence a system is generally encapsulated in two features: the estimation of the disturbance and how the estimated disturbances are incorporated in a control law. There are exceptions where the control law uses no estimated disturbance but incorporates features providing disturbance rejection indirectly. These three approaches for disturbance rejections of a multirotor is found in the literature and is described below.

2.2.1 Indirect Disturbance Rejection

The most common method of providing disturbance rejection is to use integrators in the control law. These integrators wind up to absorb any disturbance or uncertainty in the system causing it not to maintain its steady-state condition. Integrators for disturbance rejection is typically used as the baseline to compare more sophisticated techniques for disturbance rejection.

[1]Figure was created by Anton Erasmus, and is used with permission.

Another method of rejecting disturbances without the use of an estimator is the Nonlinear Dynamic Inversion (NDI) technique. NDI's control law is designed based on the assumption of fast sampling period from the Inertial Measurement Unit (IMU) sensors. The NDI technique produces a high bandwidth controller which enables quick response times from disturbances. It has been tested practically for multirotors to reject wind gust, as shown by Smeur *et al.* [9]. Other methods use adaptive control to adjust the gains of the controllers according to an update-law which will view the disturbances as changes to the physical model. These techniques require fast adaptation speed to adjust for disturbances such as wind gusts as suggested by Fernandez *et al.* [10].

Data-driven techniques have risen in popularity for the use of disturbance rejection due to the difficulty of modelling the various disturbances affecting a multirotor. Using the measurement data from previous practical flights allow these data-driven techniques to learn the nonlinear function describing these disturbances. One of these data-driven techniques is the use of neural networks with supervised learning. Supervised learning of neural networks is when the input and output data is known, and the neural network learns the nonlinear mapping using an optimiser. Celen and Oniz [11] and Al-Mahasneh *et al.* [12] trains a neural network to act as a controller of the multirotor to account for the nonlinear behaviour which the linear controller is unaware. A closely related technique known as Reinforcement Learning (RL) also replaces the classical controller entirely and learns to control the multirotor through thousands of interactions in the simulation environment. During the optimisation of the RL controller, it will start to learn how to behave when disturbances are affecting the multirotor. This is shown by Koch *et al.* [13], Vankadari *et al.* [14] and Hwangbo *et al.* [15] which uses the RL controller to control a multirotor.

2.2.2 Rejection for Specific Disturbances

Other control laws are designed with specific disturbance phenomena in mind. Matus-Vargas *et al.* [29] developed an algorithm to switch between two different controllers where one is specifically designed to reject ground effects if it is detected by the algorithm. Bannwarth *et al.* [28] focus on wind disturbances by adding a wind model during the modelling of the multirotor and allows controller gains to be designed for specific weather conditions.

Data-driven approaches have been used to combat specific disturbances affecting a multirotor. This is shown by Shi *et al.* [26] who trains a neural network from experimental data to estimate a model for the ground effects. They then use this estimation in feedback to reject the ground effects. Allison *et al.* [27] use a neural network to learn the wind velocity in which a multirotor is flying using the measurement data. This estimation can now be used to improve the flight controller.

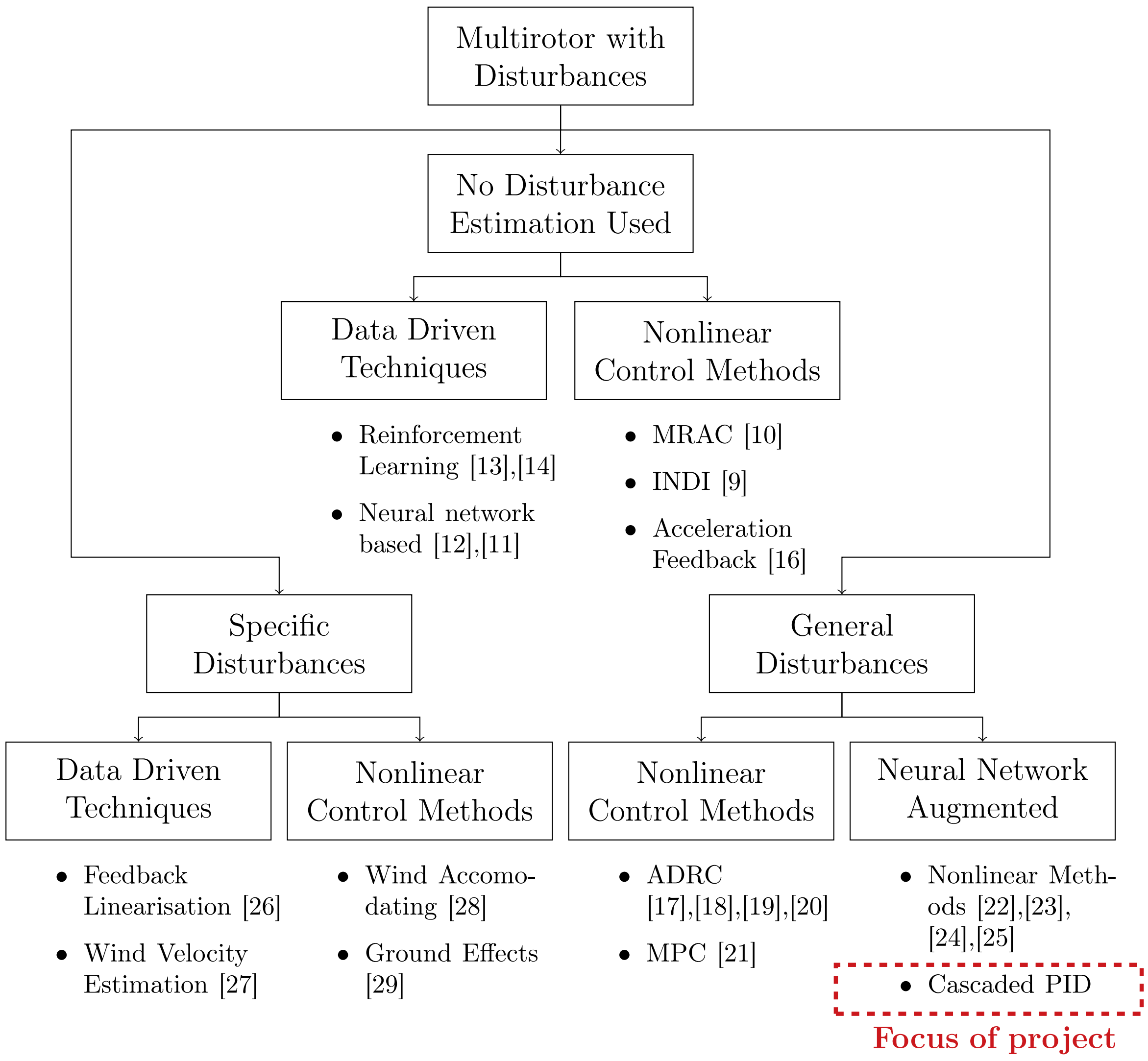

Multirotor with Disturbances
No Disturbance Estimation Used
Data Driven Techniques
• Reinforcement Learning [13],[14]
• Neural network based [12],[11]
Nonlinear Control Methods
• MRAC [10]
• INDI [9]
• Acceleration Feedback [16]
Specific Disturbances
General Disturbances
Data Driven Techniques
• Feedback Linearisation [26]
• Wind Velocity Estimation [27]
Nonlinear Control Methods
• Wind Accomo-dating [28]
• Ground Effects [29]
Nonlinear Control Methods
• ADRC [17],[18],[19],[20]
• MPC [21]
Neural Network Augmented
• Nonlinear Meth-ods [22],[23], [24],[25]
• Cascaded PID
Focus of project

Figure 2.3: Summary of the different approaches to disturbance rejection for multirotors.

2.2.3 General Disturbances

The data-driven techniques have further been introduced to assist the nonlinear controllers for the primary purpose of model uncertainty. Since dynamics are omitted during the modelling process, which the classical controllers are unaware, these data-driven techniques are used to combat them. The data-driven techniques are mostly combined with the use of a linear controller and as such is said to augment the classical controllers. The data driven techniques once again make use of neural networks and augment them in the following ways: by adapting the gains of the nonlinear controller shown by Bari *et al.* [25] or adding an additional control signal to account for disturbances shown by Jiang *et al.* [30], Verberne and Moncayo [22], Bisheban and Lee [24] and

Xiang *et al.* [23].

Similar methods exist in which the nonlinear controller is assisted by classical estimation techniques. One of these classical estimators are the Extended State Observer (ESO) and is mainly used to estimate the combined disturbances influencing a multirotor. Zhang *et al.* [18], Suhail *et al.* [19] and Zhao *et al.* [20] all use the ESO to estimate the disturbance and then uses the estimated disturbance in a control law to reject its effect. Other estimators that have been used are the Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF) which was used by Hentzen *et al.* [21] to reject disturbances by incorporating it into the control law. All of the previously mentioned authors used the estimators in a disturbance rejection architecture which subtracts the estimated disturbance from the control signal produced by the controller.

2.2.4 Summary of Literature Review

From this literature review, a tree diagram can be constructed which summarises the different branches within disturbance rejection. The tree diagram is shown in Fig. (2.3), where the project's approach to the problem definition is highlighted. This is done in order to clearly identify where the approach to the problem definition fits in the literature. The supporting arguments for arriving at the selected approach will be discussed next.

2.3 Proposed Solution

As shown in the previous section, most of the literature on disturbance rejection in multirotors focused on using mathematical models describing the disturbances, designing controllers for specific disturbance phenomena, or using various estimators. It is also evident that current approaches use data-driven techniques in which mathematical modelling appears non-tractable. The project's selected solution is based on past proposed solutions as well as the fact that data-driven approaches are currently being explored. Considering the above-mentioned approach, the solution should be:

- easily implemented on existing flight controllers
- practically feasible
- general enough to be used in a wide range of applications
- make use of data-driven approaches
- does not replace the classical controller

The literature review revealed the following:

- The design of specific control laws for disturbance phenomena is limited to the specific disturbances which are designed for and do not cater to all the various applications a multirotor could be used for: carrying payloads, inspection of surfaces and flying in unfavourable weather conditions.
- The estimators used for disturbance rejection have shown to have the means to estimate a wide class of disturbances.
- Recently there have been attempts to use data-driven techniques to estimate specific disturbance phenomena influencing multirotors.
- There is little development in creating mathematical aerodynamic models for multirotors.
- There are control laws which displayed good practical disturbance rejection, but does not make use of commercial off the shelve (COTS) flight controllers.

Taking into account these considerations, a data-driven technique which estimates the disturbance effecting a multirotor and integrating the estimated disturbance in an existing flight controller law was adopted. The data-driven technique selected makes use of neural networks to estimate the disturbance affecting a multirotor. The selection is based on the fact that neural networks have achieved numerous advancements in estimating disturbances in multirotors and controlling robots [26], [27], [31]. The use of neural networks in multirotor control systems is also relevant in the literature [30], [22], [24], [23], [12].

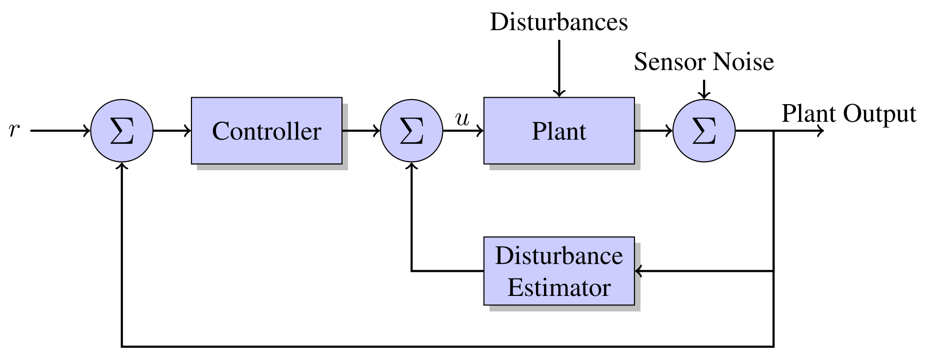


Figure 2.4: The general disturbance rejection architecture used in various applications for the rejection of external disturbances.

The estimated disturbance will be used in a disturbance rejection architecture shown in Fig. (2.4) in which the estimated disturbance is subtracted by the original control signal produced by the controller. This architecture was selected as it has been used extensively as the method to reject disturbances using an estimator [18], [19], [20].

2.4 Multirotor Overview

The multirotor is a six degree of freedom (6DoF) rigid body with four independently controlled motors equally spaced around the centre of gravity (CoG) shown in Fig. (2.5). These motors provide the forces to allow the multirotor to translate and rotate in space. The forces and moments acting on the multirotor with respect to its acceleration, velocity and position are called the kinetic equations and are derived using Newton's second law. This results in

$$\begin{aligned} \boldsymbol{F}_{\mathcal{B}} &= m\dot{\boldsymbol{V}}_{\mathcal{B}} + \boldsymbol{\Omega}_{\mathcal{B}} \times m\boldsymbol{V}_{\mathcal{B}} \\ \boldsymbol{M}_{\mathcal{B}} &= \boldsymbol{I}\dot{\boldsymbol{\Omega}}_{\mathcal{B}} + \boldsymbol{\Omega}_{\mathcal{B}} \times \boldsymbol{I}\boldsymbol{\Omega}_{\mathcal{B}} \end{aligned} \tag{2.1}$$

where

$$\begin{aligned} \boldsymbol{F}_{\mathcal{B}} &= [F_{\mathcal{B}_x}, F_{\mathcal{B}_y}, F_{\mathcal{B}_z}]^\top \\ \boldsymbol{M}_{\mathcal{B}} &= [M_{\mathcal{B}_x}, M_{\mathcal{B}_y}, M_{\mathcal{B}_z}]^\top \end{aligned} \tag{2.2}$$

are the forces and moments in the various body directions on the multirotor. The forces and moments acting on the multirotor are

$$\begin{aligned} \boldsymbol{F}_{\mathcal{B}} &= \boldsymbol{F}_{\mathcal{B}}^T + \boldsymbol{F}_{\mathcal{B}}^G \\ \boldsymbol{M}_{\mathcal{B}} &= \boldsymbol{M}_{\mathcal{B}}^T + \boldsymbol{M}_{\mathcal{B}}^G \end{aligned} \tag{2.3}$$

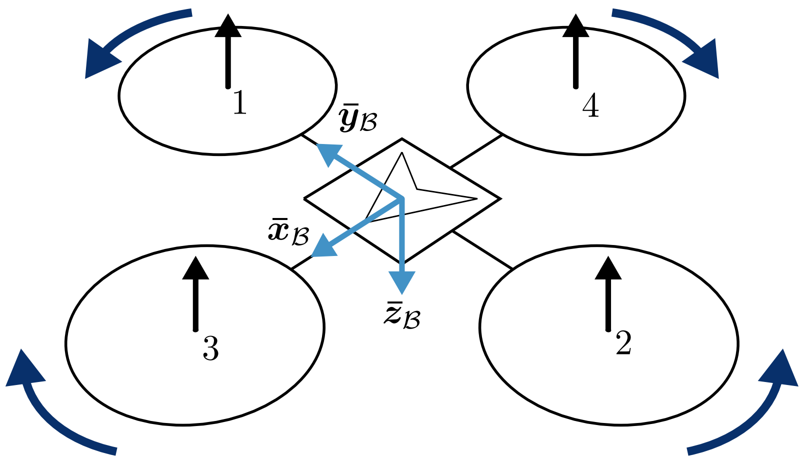

1
4
3
2
$\bar{\boldsymbol{y}}_{\mathcal{B}}$
$\bar{\boldsymbol{x}}_{\mathcal{B}}$
$\bar{\boldsymbol{z}}_{\mathcal{B}}$

Figure 2.5: Illustration of a multirotor.

where the superscripts T and G refer to the thrust produced by the motors and gravity respectively.

The linear velocity and angular velocity of the multirotor is given by

$$\begin{aligned} \boldsymbol{V}_{\mathcal{B}} &= [V_{\mathcal{B}_x}, V_{\mathcal{B}_y}, V_{\mathcal{B}_z}]^\top \\ \boldsymbol{\Omega}_{\mathcal{B}} &= [\Omega_{\mathcal{B}_x}, \Omega_{\mathcal{B}_y}, \Omega_{\mathcal{B}_z}]^\top. \end{aligned} \tag{2.4}$$

The mass of the multirotor is given by m and

$$\boldsymbol{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{zy} & I_{zz} \end{bmatrix} \tag{2.5}$$

is the moment of inertia matrix of the multirotor. The diagonal entries of the matrix are known as the principal moment of inertia. The off-diagonal entries are known as the products of inertia and assumed to be zero since the multirotor is symmetric. The rotation of each motor creates a torque in the $\bar{\boldsymbol{z}}_{\mathcal{B}}$-axis, and the combined torque is zero by rotating motor one and two in the opposite direction of motor three and four. The multirotor hovers by producing the same thrust by all four motors and changes its altitude by increasing or decreasing the thrust produced by each motor equally. Translation is achieved by performing a pitch or roll manoeuvre in the correct direction. Accelerating in the $\bar{\boldsymbol{x}}_{\mathcal{B}}$ direction, the multirotor must pitch, which corresponds rotating around the $\bar{\boldsymbol{y}}_{\mathcal{B}}$-axis. Rotation around this axis is achieved by increasing the thrust produced by motor three and decreasing the thrust produced by motor 4. A roll manoeuvre is a rotation around the $\bar{\boldsymbol{x}}_{\mathcal{B}}$-axis and results in the translation in the $\bar{\boldsymbol{y}}_{\mathcal{B}}$-axis and is achieved by increasing the thrust produced by motor two by the amount motor one is decreased. Yawing corresponds to the rotation around the $\bar{\boldsymbol{z}}_{\mathcal{B}}$-axis and is done by increasing the thrust of motors

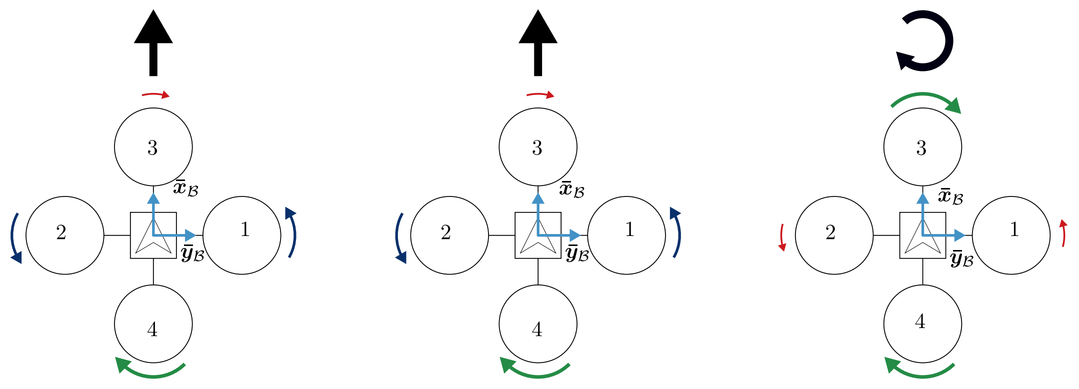

3
2
1
4
$\bar{\boldsymbol{x}}_{\mathcal{B}}$
$\bar{\boldsymbol{y}}_{\mathcal{B}}$
3
2
1
4
$\bar{\boldsymbol{x}}_{\mathcal{B}}$
$\bar{\boldsymbol{y}}_{\mathcal{B}}$
3
2
1
4
$\bar{\boldsymbol{x}}_{\mathcal{B}}$
$\bar{\boldsymbol{y}}_{\mathcal{B}}$

Figure 2.6: The various manoeuvres that the multirotor is capable of doing based on the increased and decreased thrust produced by the correct motors.

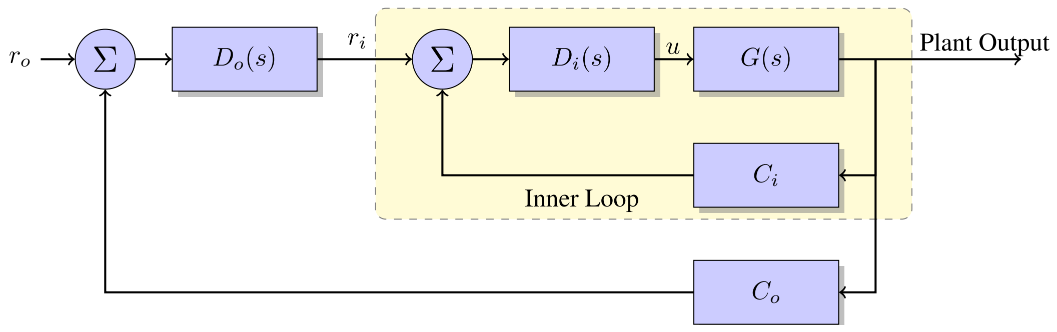


Figure 2.7: The succesive loop closure control architecture.

three and four and decreasing the thrust of motors one and two by the same amount. The above-mentioned manoeuvres are depicted in Fig. (2.6).

2.5 Successive Loop Closure Control

The control system responsible for stable translation and rotation of a multirotor is designed using the linear model of the multirotor. The two equations in 2.1 are both nonlinear equations and using taylor series expansions around the hover condition results in the linear model of the multirotor. From the linear model, a control system is designed responsible for stable translation and rotation of the multirotor.

By analysing the eigenvalues of the set of linear equations, the multirotor dynamics are separated from fast to much slower dynamics. This time separation in dynamics is intuitively understood by the fact that the multirotor must first pitch or roll before it starts to translate in the desired direction. It follows that the dynamics of the multirotor are ordered from fastest to slowest as: angular rate, angular, velocity and then position.

The control systems designed for the multirotor exploit this phenomenon where feedback control loop after feedback control loop is closed, with each loop abstracting the multirotor dynamics from fastest to slowest as each loop is closed. For multirotors, these feedback loops are called the angular rate, angle, velocity, and position loops. Fig. (2.7) depicts the successive loop closure control technique where the inner-most controller, $D_i(s)$ acts directly on the multirotor and is designed using the linear model. The addition of the inner controller results in the changing of the model's dynamics. The model which the outer controller observes is the model whose dynamics have been changed by the inner controller and not the original plant. The outer controller acts on the inner loop containing the inner-controller and the multirotor's linear model. Thus, the outer controller generates the reference for the inner controller to follow. The $\boldsymbol{C}$ matrices extract the state of interest for the controller to control and will be the angular rate for the inner controller and angle for the outer controller.

2.6 Quaternions

Euler angles are most commonly used to describe the attitude of a vehicle relative to a fixed axis shown in Fig. (2.8). This fixed axis is commonly known as the North-East-Down (NED) axis where the x-axis points to North and y-axis to East. There exists a matrix known as the Direct Cosine Matrix (DCM) which converts between these two axes and is used extensively during a flight of a multirotor. Using Euler Angles to represent the attitude of the vehicle leads to singularities at specific attitudes and are cumbersome for controllers. To overcome these singularities unit quaternions are used to represent the attitude of the vehicle, which is shown in Fig. (2.9).

Unit quaternions are a four-dimensional description, $\boldsymbol{q} = [q_0, q_1, q_2, q_3]$, of three-dimensional rotations. It is free of singularities and computationally efficient. In the case of small angles the following relationships exist between Euler angles and unit quaternions:

$$\begin{aligned} \phi &= -2q_1, \\ \theta &= -2q_2, \\ \psi &= -2q_3, \text{ and} \\ |\boldsymbol{q}| = 1 &= \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}. \end{aligned} \tag{2.6}$$

For an in-depth understanding of unit quaternions beyond small-angle approximation, refer to [32].

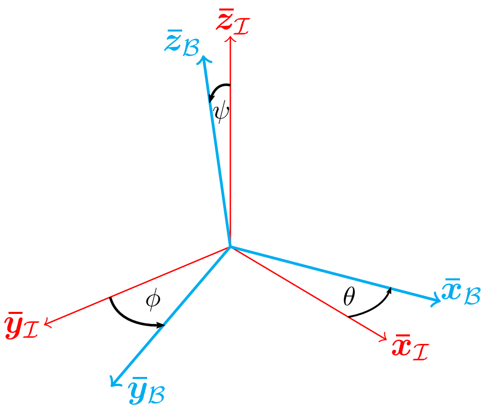


Figure 2.8: Euler angle representation between the fixed axis, $\mathcal{I}$ and the body axis $\mathcal{B}$.

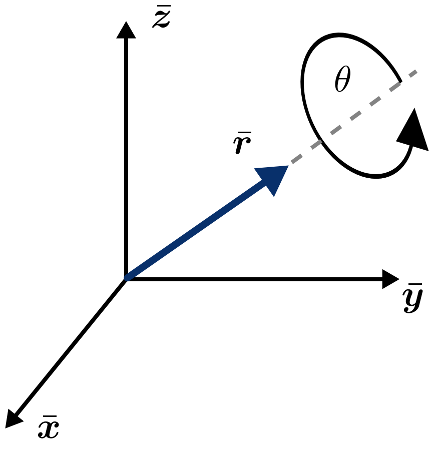


Figure 2.9: Unit qauternion representation of a rotation.

2.7 Neural Networks

Neural networks (NN) are a nonlinear modelling tool used to associate input and output patterns with the use of learning algorithms. The neural network thus learns a function which approximates this association between the input and output. There are many different neural network architectures, and the project will only be focusing on two different types: feedforward and recurrent neural networks.

2.7.1 Feedforward Neural Network

Feedforward neural networks receive its name from the flow of data through the network: the input data flows forward through the network undergoing intermediate computations before being outputted. There are no feedback loops which feeds the outputs of the network back [33].

The feedforward NN consists of an input layer, a hidden or multiple hidden layers and an output layer. Each layer contains units which perform a computational operation. One of these units can be seen in Fig. (2.10), which is in the exploded view. Each unit in a layer, l, is connected to all of the units in the following layer where each connection has its own weight, $w_{jk}^{(l)}$, and each unit has its own bias, $w_j^{(l)}$. jk refers to the connection between a unit j and unit k whereby unit k is in the proceeding layer of unit j.

These units perform a nonlinear computational operation taking multiple inputs and output a single value. The computational operation sums all the incoming connections, adds the unit's bias value and then pushes this summa-

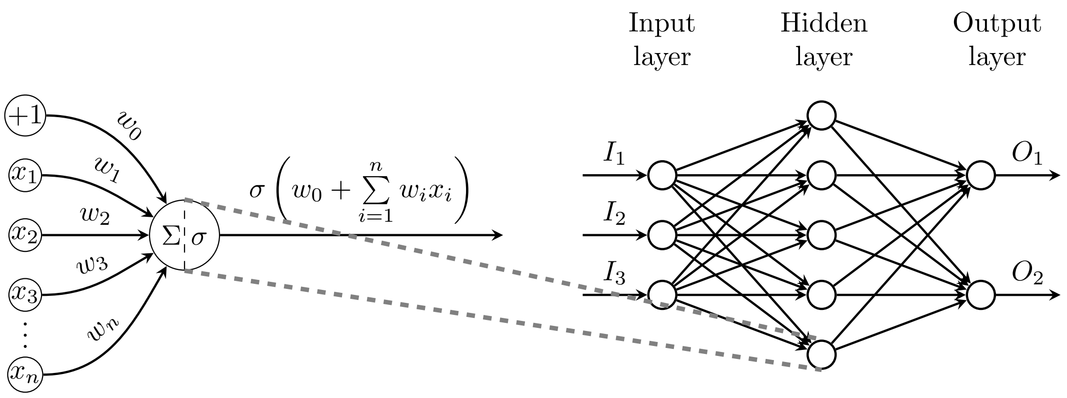


Figure 2.10: Feedforward neural network with one hidden layer with a single unit in a exploded view [1].

tion through an activation function as shown by

$$x_k^{(l+1)} = \sigma(w_j^{(l)} + \sum w_{jk}^{(l)} \cdot x_j^{(l)}). \tag{2.7}$$

Before the unit can perform its computational operation, the output of each unit in the previous layer is multiplied by their corresponding connection's weight before arriving at the unit.

The ability to learn nonlinear behaviour is enabled by the fact that the activation functions are nonlinear. By increasing the number of units through additional layers or increasing the number of units in a layer increases the ability of the neural network to learn more complex behaviour. The amount of complexity that neural networks can learn refers to the capacity of the neural network and is an open problem to find quantitative methods to estimate.

2.7.2 Long Short Term Memory (LSTM)

Challenges for feedforward NNs are time series based data where long term dependencies exist and influence the next state of the system. To address this short coming of feedforward NNs, recurrent neural networks (RNN) were implemented, which contains loops to retain information as shown in Fig. (2.11). An RNN can be imagined as multiple neural network architectures being repeated with each passing a message to its successor. However, in theory, RNN is capable of learning long term dependencies, but in practice, they do not and is explained in Bengio *et al.* [34]. Many improvements have been made with RNNs, and one of these improvements is Long Short Term Memory (LSTM) units develop by Hochreiter and Schmidhuber [35].

LSTM units are a type of recurrent neural network having three inputs and two outputs. This is seen in Fig. (2.12) where a layer of multiple LSTM units are shown with a single LSTM unit in the exploded view. The LSTM improves

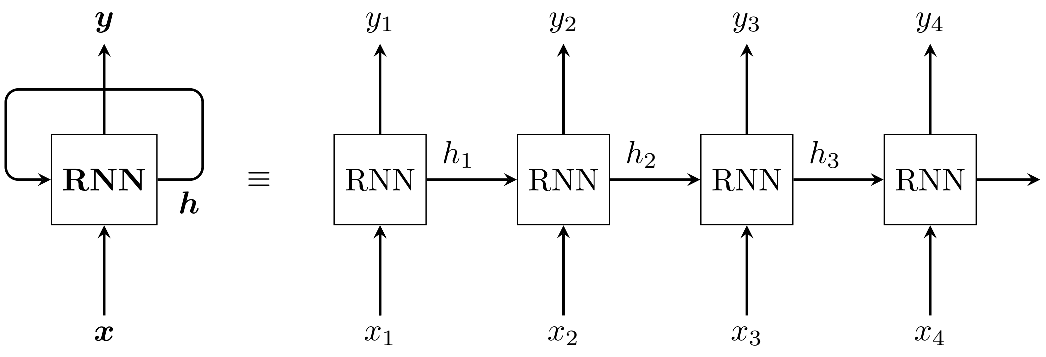


Figure 2.11: A unrolled RNN.

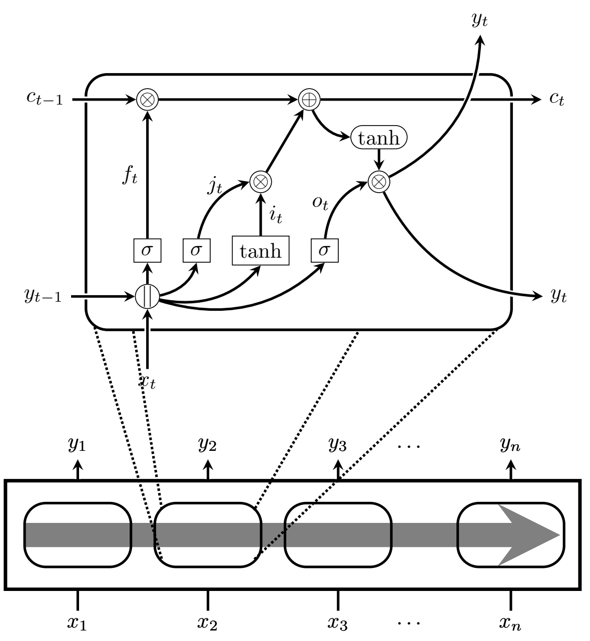


Figure 2.12: A layer containing LSTM units with one of the units presented in a exploded view [1].

significantly on feedforward neural networks with time series based data which contain long-term dependencies [33].

Within an LSTM unit, there are multiple operations occurring, executing like a conveyor belt. The first step in the LSTM unit is the *forget gate* which determines what information is going to be thrown away. This operation corresponds to

$$\boldsymbol{f}_t = \sigma(W_\mathrm{f}[\boldsymbol{y}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_\mathrm{f}), \tag{2.8}$$

which is a single feedforward layer with two inputs: the output of the previous LSTM unit in the layer, $\boldsymbol{y}_{t-1}$, and the input to the current LSTM unit, $\boldsymbol{x}_t$. The next step of the LSTM is to determine what information should be stored. This is shown by the *input gate layer* and corresponds to

$$\boldsymbol{j}_t = \sigma(W_\mathrm{x}[\boldsymbol{y}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_\mathrm{x}), \tag{2.9}$$

which again is a single feedforward layer using the sigmoid activation function. Following this operation, is the single feedforward layer using the tanh activation function. This layer produces a list of possible states that could be remembered. This operation corresponds to:

$$\boldsymbol{i}_t = \tanh(W_\mathrm{c}[\boldsymbol{y}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_\mathrm{c}). \tag{2.10}$$

The LSTM unit state is $\boldsymbol{c}_t$, and this is updated by using the previous mentioned results:

$$\boldsymbol{c}_t = \boldsymbol{f}_t \times \boldsymbol{c}_{t-1} + \boldsymbol{j}_t \times \boldsymbol{i}_t. \tag{2.11}$$

This operation can be described as updating the LSTM unit state by removing the information that the LSTM believes should be forgotten and then adding what the LSTM believes should be remembered from the current input.

The final step is the output and is based on the updated LSTM state and inputs. This operation is described as:

$$\begin{aligned}\boldsymbol{o}_t &= \sigma(W_\text{o}[\boldsymbol{y}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_\text{o}) \\ \boldsymbol{y}_t &= \boldsymbol{o}_t \tanh(\boldsymbol{c}_t).\end{aligned} \tag{2.12}$$

This output is then fed to the following LSTM unit in the layer, where the entire operation is repeated.

These multiple operations in a recurrent neural network come with the cost of significantly increased training time. There are also multiple variants of recurrent neural networks such as the Gated Recurrent Unit (GRU).

2.7.3 Activation Functions

The activation function is the operation the layers' units perform after the summation. There exists a number of different activation functions, and the correct choice is based on the type of problem and type of neural network architecture that is being used.

Activation functions are nonlinear functions which are generally continuous everywhere. This characteristic of being smooth allows efficient and quick calculation of the gradient. Fig. (2.13) shows some of the common activation functions seen in the literature.

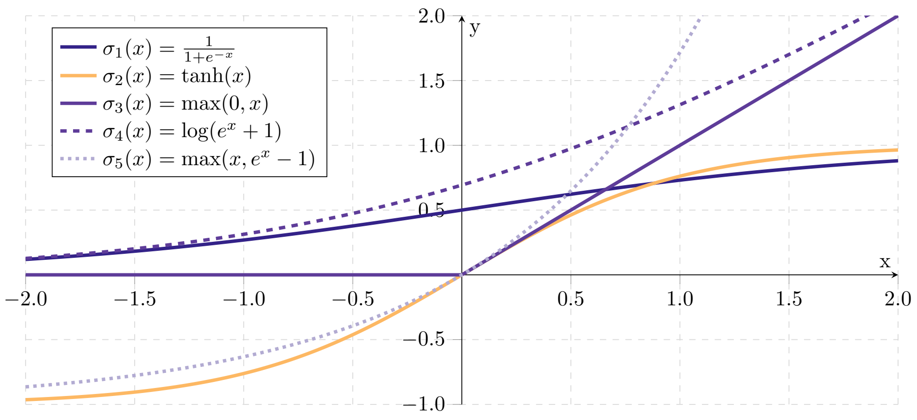


Figure 2.13: Various activation functions found in literature [1].

2.7.4 Loss Function

The loss function is the function by which the neural network is scored on how well it is performing. There are different loss functions available to choose from, depending on the type of problem. These are the mean squared error (MSE) seen in Equation 2.13 and cross entropy seen in Equation 2.14.

$$J_0 = \frac{1}{n}\sum_{i=0}^{n} \mid \boldsymbol{\hat{y}_i} - \boldsymbol{y_i} \mid \tag{2.13}$$

$$J_0 = \frac{1}{n}\sum_{i=0}^{n} \boldsymbol{y_i} \cdot \log(\boldsymbol{\hat{y}_i}) \tag{2.14}$$

MSE is the most commonly used loss function for time series based data, whereas cross entropy is more commonly used for classification.

2.7.5 Gradient-Based Learning

The nonlinearity of neural networks causes the loss function to become nonconvex, resulting in them being trained using iterative, gradient based optimisers. These optimisers try to minimise the loss function to a very low value [33]. These optimisers are the algorithms that update the trainable variables in the neural network based on the effect they had on the loss function. This effect is determined by computing the gradient of the trainable variable with respect to the loss function as shown by

$$\frac{\partial J_0}{\partial W_{\mathrm{i,j}}} = \frac{\partial J_0}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial W_{\mathrm{i,j}}}. \tag{2.15}$$

Choice of optimiser varies from problem to problem, and the decision is based on literature, but common optimisers are Adam, Adagrad and Adadelta.

2.7.6 Underfitting And Overfitting

The goal of the neural network is to perform well on unseen data that was not part of the training dataset, and this is tested by a validation dataset during training. As the neural network trains on the training dataset, it should improve on both validation and training dataset. In the region where the neural network improves on both the validation and training dataset, the neural network has underfit the dataset, and more training is required. However, there is a point during training where the loss function on the validation dataset increases signifying overfitting. Overfitting indicates that the neural network has stopped learning the correlation in the data and has started to become a lookup table for the training dataset and outputs uncorrelated answers for anything outside of the training dataset. The ability of a neural network to perform well on unseen data is characterised as generalisation [33].

Fig. (2.14) shows the different regimes during training. On the left, the generalisation error and training error is high, indicating the neural network still requires training, however as the neural network converges it reaches an optimal point before starting to overfit the training data. In the overfitting regime, the training error is small; however, the neural network does not generalise well and has overfitted the training data.

2.7.7 Hyperparameters

During the training of a neural network, there are untrainable variables which are selected by the individual. The individual has complete control of these parameters, and they influence the training results. They affect the generalisation of the model, training error and computational resources [33].

These hyperparameters are the learning rate, weight regularisation coefficient, dropout and number of hidden units. These parameters are tuned iteratively to determine which combinations result in the lowest loss function.

Learning Rate

The learning rate is commonly notated as α and refers to the step size the optimiser may take in the direction which minimises the loss function. Increasing the learning rate too much could lead to increasing the loss function. Decreasing it too much will lead to slower training, but may cause the system to converge to an unacceptable large loss function [33]. The effect of the

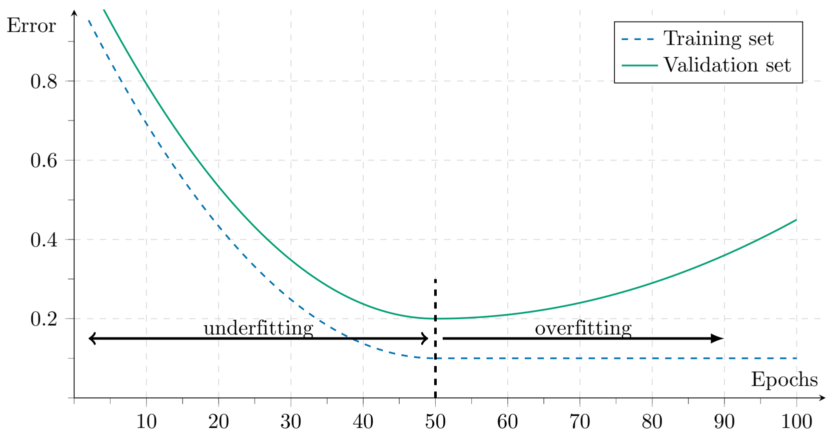


Figure 2.14: The regions of underfitting and overfitting during training [1].

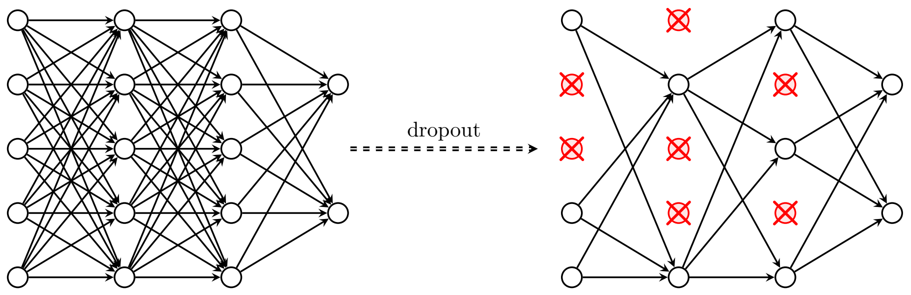


Figure 2.15: The result of dropout on the architecture of a neural network [1].

learning rate can be seen in the following equation,

$$w_t = w_{t-1} - \alpha \frac{\hat{m}}{\sqrt{\hat{v}_t} - \epsilon}, \tag{2.16}$$

which uses the Adam optimiser where the variables $\hat{m}$, $\hat{v}_t$ and ϵ are all a function of the gradient of the weight, w, with respect to the loss function.

Weight Regularisation

A method known for improving the generalisation of a trained model is forcing the weights to be as small as possible [36]. This is achievable by adjusting the loss function to include the size of the weights as shown

$$J(\boldsymbol{\theta}) = J_0(\boldsymbol{\theta}) + \frac{1}{2}\lambda \sum_i w_i^2. \tag{2.17}$$

J_0 would be the original chosen loss function such as MSE and λ the weight regularisation coefficient.

Increasing the weight regularisation coefficient will result in the optimiser to punish larger weight values, and decreasing it would allow weights to be larger.

Dropout

Dropout is another method of improving the generalisation of a neural network. It refers to temporary dropping out units from the neural network, as seen in Fig. (2.15) during a single training sample. The units are selected from a fixed probability, p, independent from the other units.

The selection of this probability, p, is advised to be between 0.4 and 0.5 [37]. Making this value too large will result in reduced training results, and making it too small in possible overfitting of the training data.

2.8 Summary

This chapter explained the origin of disturbances in systems and how they are classified to provide the necessary understanding of why disturbances arise in systems. This was followed by a literature study on methods which are used to reject disturbances influencing a multirotor. The literature provided the knowledge to support the proposed solution to the problem definition, which was presented. Following the proposed solution, which is to reject disturbances using a neural network in a disturbance rejection architecture, the necessary technical background to understand the concepts and terminology used in the project was presented. This technical background included concepts about control systems and machine learning.

Chapter 3

System Overview

This chapter describes the various components required for implementing a neural network for rejection disturbances on a multirotor. The interdependency of the components is described as well as the workflow from start to end. These components include the control system architecture and software responsible for stable flight, the simulation environment for validation and data generation, the neural network application programming interface (API) and the communication layer for resolving interdependency between components.

3.1 PX4 Software Stack

PX4[1] is an open source flight control eco-system providing software from the low level firmware up to the user interface for waypoint flying. The PX4 firmware comprises out of various modules that communicate with each other using an asynchronous publish-subscribe architecture. PX4 runs on the NuttX

[1]https://px4.io/

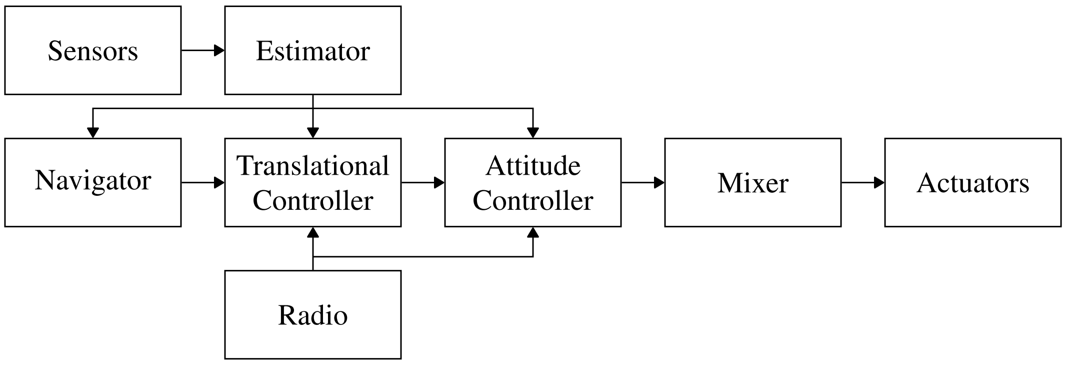


Figure 3.1: PX4 firmware consists out of various modules with arrows indicating communication direction.

State
Quaternions
Velocity in NED-frame
Position in NED-frame
Gyroscope delta angles bias
Accelerometer bias
Earth Magnetic Field Vector
Magnetometer bias errors
Wind Velocity

Table 3.1: The states being estimated by the PX4 EKF.

real time operating system (RTOS) providing real time, deterministic and priority execution of services. This allows the PX4 source code to be built within a Linux system which is emulating the NuttX RTOS and allows simulations to run with the same code as on the embedded system. The project will mainly be using the firmware of PX4, which is responsible for executing the control laws allowing the multirotor to hover and fly as desired. A block diagram of the various modules is shown in Fig. (3.1) with the arrows indicating communication directions, and it is only required to understand the following modules: estimator, translational controller and the attitude controller.

3.1.1 Estimator

PX4 implements a kinematic EKF, receiving measurements from the various sensors and combines them to provide an estimate of the multirotor states. These states are shown in Table 3.1 and should be noted that the PX4 EKF estimates sensor biases for the gyroscope, accelerometer and magnetometer. The reasoning for highlighting this fact will become clearer in Chapter 5. The update rate of the PX4 EKF is 1kHz and publishes the estimated states at 250Hz.

3.1.2 Translation and Attitude Controller

PX4 uses a cascaded control architecture containing an inner attitude and an outer translation controller, as shown in Fig. (3.2). The attitude controller consists of the angular rate controller, which uses a nonlinear Proportional-Integrated-Derivative (PID) control law shown in Fig. (3.3) and a nonlinear proportional control law for the angle controller. The translation controller follows the same convention with a nonlinear PID controller for the velocity loop and a proportional controller for the position. The nonlinear PID controller can be simplified to the standard PID control law and is also used to

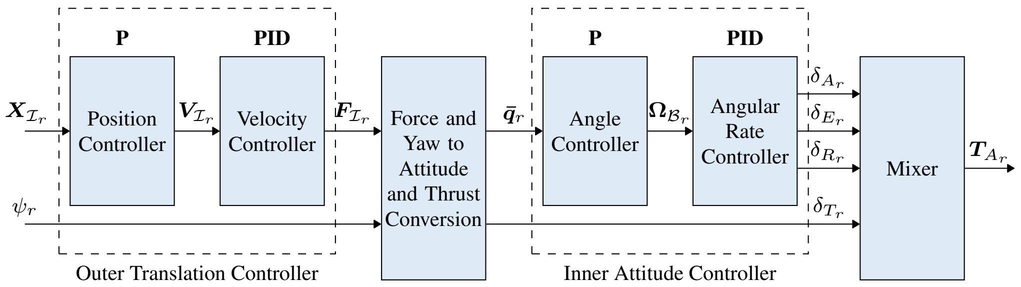

P
PID
Position Controller
Velocity Controller
Force and Yaw to Attitude and Thrust Conversion
P
PID
Angle Controller
Angular Rate Controller
Mixer
$\boldsymbol{X}_{\mathcal{I}_r}$
$\boldsymbol{V}_{\mathcal{I}_r}$
$\boldsymbol{F}_{\mathcal{I}_r}$
$\bar{\boldsymbol{q}}_r$
$\boldsymbol{\Omega}_{\mathcal{B}_r}$
δ_{A_r}
δ_{E_r}
δ_{R_r}
δ_{T_r}
$\boldsymbol{T}_{A_r}$
ψ_r
Outer Translation Controller
Inner Attitude Controller

Figure 3.2: PX4 control architecture used for controlling a multirotor.

design the controller gains,

$$\delta_{\text{virtual}} = P(\Omega_{\mathcal{B}_{\text{i,r}}} - \Omega_{\mathcal{B}_{\text{i}}}) - I\int(\Omega_{\mathcal{B}_{\text{i,r}}} - \Omega_{\mathcal{B}_{\text{i}}}) + D\frac{d}{dt}(\Omega_{\mathcal{B}_{\text{i,r}}} - \Omega_{\mathcal{B}_{\text{i}}}) \tag{3.1}$$

with $\Omega_{\mathcal{B}_{\text{i,r}}}$ representing the desired body angular rate of the multirotor in the i direction and $\Omega_{\mathcal{B}_{\text{i}}}$ the measured angular rate of the multirotor. The angular rate controller outputs a virtual control signal, δ_{virtual}, which is represented by three virtual surface deflections adopted by aeroplanes representing the desired change in pitch, roll and yaw. The mixer block is responsible for translating the virtual control signals to the desired thrust produced by each motor. The same PID control law in Equation 3.1 is used for the velocity controller by replacing $\Omega_{\mathcal{B}_{i,r}}$ and $\Omega_{\mathcal{B}_i}$ with $V_{\mathcal{I}_{i,r}}$ and $V_{\mathcal{I}_i}$, respectively. The controller block *Force and Yaw to Attitude and Thrust Conversion* convert the force setpoint that the velocity controller produces to a quaternion and thrust setpoint. This conversion can be read more at [38].

The linearised proportional controller for the angle and inertial position in the various directions, is in the form

$$\dot{Z}_{\text{i,r}} = P(Z_{\text{i,r}} - Z_{\text{i}}), \tag{3.2}$$

with $\boldsymbol{Z}$ representing a placeholder for either the inertial position, $\boldsymbol{X}_{\mathcal{I}}$, or quaternions, $\boldsymbol{q}$, of the multirotor.

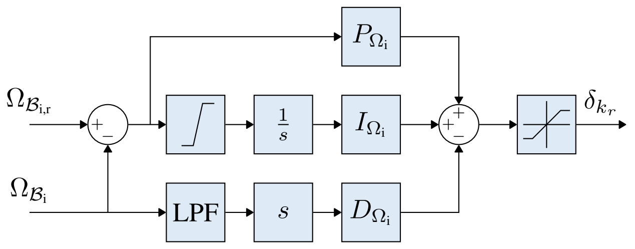


Figure 3.3: The PX4 angular rate control blockdiagram showing a PID controller with additional elements for practical flight considerations.

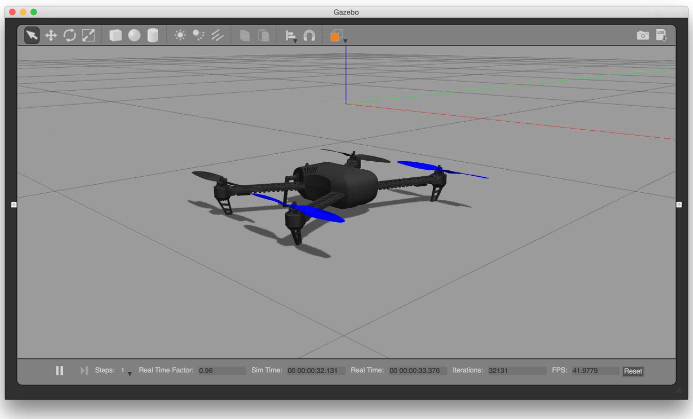


Figure 3.4: Gazebo simulating the IRIS multirotor alongside PX4 [2].

The design of the P, I and D gains are determined using a Root Locus for each of the different controllers in each direction. During the design process, it is essential that each closed-loop system, starting from the inner-most loop, is consecutively separated by sufficient bandwidth. This is to ensure that the controllers do not compete with each other, resulting in an oscillatory response. The design rule-of-thumb is between 5-10 times slower than the inner closed-loop system's cut-off frequency. The other components in Fig. (3.3) are implemented for practical flight considerations such as the low pass filter (LPF), saturation block and integral limiter. The reasons for their implementations and the design process of determining the gains can be read more at [32].

3.2 Gazebo Physics Simulation

Gazebo is a 3D dynamic multi-robot environment capable of approximating the real world in which robots operate. Fig. (3.4) shows the Gazebo simulation environment in which a multirotor is spawned. Gazebo makes use of the Open Dynamics Engine to simulate rigid body dynamics and include noise models for the various sensors used on a multirotor such as IMU, Global Positioning System (GPS), barometer, and magnetometer. These noise models include sensor bias, random walk and high frequency noise which can be adjusted to represent the physical multirotor noise profiles. Gazebo also includes the nonlinear models for thrust produced by motors rotating a propeller.

Gazebo allows the simulation environment to be enriched with the use of so-called plugins. Plugins can be written to provide additional functionality to the simulation environment. This allows models to interact with their environment or exhibit different behaviour or appearances. The specific plugin and its purpose will be introduced in Chapter 4.

Gazebo is widely used as the preferred simulation environment for open source projects with ample support for the PX4 development environment, and for this reasons, it was selected as the simulation environment to be used.

3.3 Robotic Operating System (ROS)

Robotic Operating System (ROS) is used in various robotics applications as the communication layer between various subsystems. ROS has branched into two versions, known as ROS1 and ROS2. ROS1 is used throughout this project and will be from now on referred to as ROS. ROS adopts a publish-subscribe architecture with a centralised point known as the ROS Master as shown in Fig. (3.5). A ROS node represents a subsystem typically fulfilling a single task. Before any other ROS nodes are able to communicate with other nodes, it must first register with the ROS master. Collectively, the ROS nodes cooperate with each other in order to complete the larger task at hand.

ROS is used to provide setpoints for the multirotor to fly in Gazebo and change the environment and physical properties of the multirotor given specific states of the system. These ROS nodes are the *trajectory generator* and *domain randomisation* ROS nodes, respectively.

3.3.1 Trajectory Generator ROS Node

A ROS node was required to provide position and velocity setpoints for the multirotor to fly during the entire simulated flight. The ROS node generates the take-off commands, then provides a random setpoint for the multirotor to reach for a predetermined time and then command the landing of the vehicle. The generation of the random setpoint is explained in detail in Chapter 4.

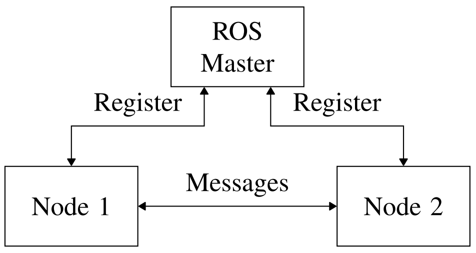


Figure 3.5: The ROS architecture makes use of a centralised node, the ROS master, which is responsible for all communications between nodes.

3.3.2 Domain Randomisation ROS Node

The multirotor flies a random trajectory in simulation and hence the landing location of the multirotor will be unknown until the land command is given. The *Domain Randomisation* ROS node is responsible for generating a large landing area for the multirotor in the Gazebo environment to allow the simulated flight test to be completed.

The ROS node is also responsible for randomising the Gazebo environment and the multirotor's physical parameters. This is done in the beginning before take-off in each simulated flight test. The reason for the randomisation of the environment and the multirotor is explained in more detail in Chapter 4.

3.4 TensorFlow

The training and the construction of various neural network architectures are done using the TensorFlow API [39]. TensorFlow provides all the necessary building blocks to train and validate the results from a neural network. This includes the mathematical backend to compute the gradients from a selected loss function, update the weights using a selected optimiser and viewing the results graphically.

3.5 MATLAB

Executing a practical flight test consists out of smaller incremental tests which become sequentially more rigours. The tests validate the selected solution in different settings which is expected to arise during a practical flight test, and the successful completion of these tests results in increased confidence when the chosen solution is practically tested.

The first test is to implement the chosen solution in a simulated environment which isolates the solution from external influences. These external influences include, but are not limited to, uncertainty in system parameters, measurement noise, computational delays and disturbances. If the chosen solution successfully meets the desired outcomes of this test, the follow-on test is known as software-in-the-loop (SITL) testing. SITL testing validates the custom software written to execute the chosen solution and contains external influences such as measurement noise.

The selected solution and classical alternatives are tested in a MATLAB environment which simulates the PX4 controllers used on the hardware. This acts as the first test of validating the selected solution towards practical flight tests. Furthermore, the MATLAB environment enables objective comparisons

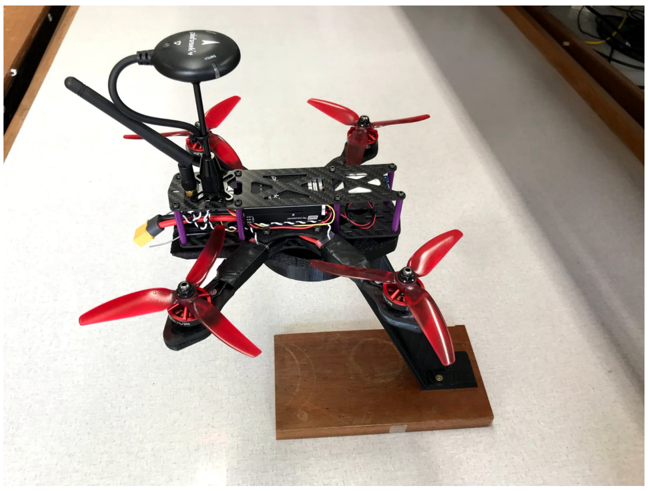


Figure 3.6: *Honeybee*, the multirotor which is simulated and used for test flights.

Mechanical property	Value
Mass	0.66 kg
Principle moment of inertia $(I_{\text{xx}}, I_{\text{yy}}, I_{\text{zz}})$	$(1.326, 0.93, 1.95) \cdot 10^{-3}$
Maximum thrust from single motor	6.47 N
Distance from CoG to motor	0.11 m
Virtual yaw moment arm	$7.997 \cdot 10^{-3}$ m
Motor time constant	0.002 s

Table 3.2: Mechanical properties of *Honeybee*.

to be done regarding specific characteristics and allows early identification of future problems.

3.6 *Honeybee*, the Multirotor

The multirotor being used throughout the project is a quadcopter with the name *Honeybee* and is shown in Fig. (3.6). It was built from off-the-shelves components and used a multirotor racing frame. The mechanical properties of *Honeybee* are shown in Table 3.2 and were calculated by a different project which coincided. The methods used for finding these mechanical properties can be found in [32]. These properties are used to simulate the multirotor and design the controllers.

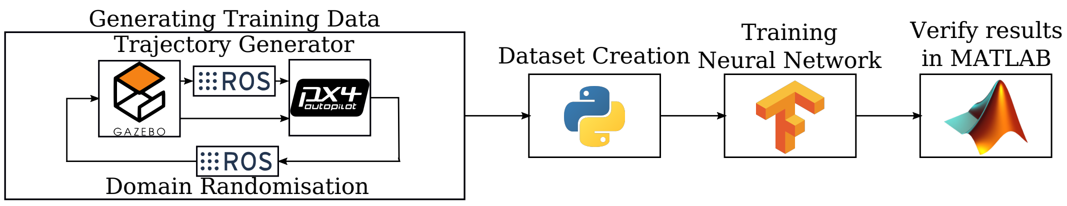


Figure 3.7: The workflow of the project containing the various components used to implement the proposed solution.

3.7 Workflow of Project

The proposed workflow of the project is shown in Fig. (3.7), which contains the components explained in this chapter and illustrates the interdependence of the various subsystems. It begins by generating the training data required for creating a dataset on which the neural networks trains. This training dataset generation occurs in the Gazebo simulation, where both ROS nodes are used to implement the mentioned techniques. The training data is processed, and a dataset is generated with the correct input and output pairs which are further elaborated in Chapter 5. Once the dataset is ready, the training of the neural networks can start using the TensorFlow API. Finally, when a satisfactory result is achieved, it can be tested in a MATLAB environment. In the MATLAB environment, results are generated, which shows the performance of the neural network as well as other commonly used techniques in a disturbance rejection architecture.

3.8 Summary

This chapter described the various components required to implement the necessary techniques to successfully train and validate a neural network for disturbance rejection on a multirotor. These components include the simulation environment for data generation, the flight control architecture for stable flight, the neural network programming interface and finally, the communication layer for components to communicate with each other. The chapter also described the workflow, which uses each component to complete the task at hand.

Chapter 4

Data Generation for Sim2Real

This chapter describes how simulated data is generated for training a neural network to estimate disturbances in reality. This process of data generation begins by adapting the simulation environment and the activities within the simulation such as the reference signal that the multirotor must follow and the disturbances influencing the multirotor. This is followed by the dataset generation and the neural network architecture for the successful training of a neural network operating in reality.

4.1 Generating Training Data

The success of a trained neural network is determined by its performance on unseen data. That is the data which the neural network has not been trained on, but comes from the same distribution the training data is sampled from. There are various techniques which impede the ability of a neural network to overfit on the training dataset, such as weight regularisation, dropout and some data manipulation techniques, but the best technique is to increase the size of the dataset. However, data in robotics is scarce and creating an experimental setup is expensive; thus, data generation does not scale well in robotics. This has caused engineers to first test and verify their newly designed systems in simulation first before, starting the process of manufacturing.

This simulation based design has birthed the idea of generating training data in a simulation environment and testing whether the neural network still performs to an acceptable level in reality. This process of generating a solution from simulation and then taking it to reality is known as Sim2Real transfer [40]. In this project, the generation of training data will occur in the Gazebo simulation environment, in which the data will henceforth be used to train a neural network using TensorFlow.

Parameter	Scaling factor range	Additive term range
mass	uniform([0.95,1.05])	-
principles of inertia of multirotor	uniform([0.95,1.05])	-
products of inertia of multirotor	-	uniform([0,0.0005])
gravity vector($\bar{\boldsymbol{x}}_{\mathcal{I}}$,$\bar{\boldsymbol{y}}_{\mathcal{I}}$,$\bar{\boldsymbol{z}}_{\mathcal{I}}$)	-	$\mathcal{N}(0, 0.2)$

Table 4.1: Ranges of parameters randomised during each simulated flight.

4.1.1 Domain Randomisation

A technique used for improving Sim2Real transfer is domain randomisation. Domain randomisation is randomising the parameters describing the simulation environment and by doing so, demands the neural network to learn multiple environments. This leads to a neural network which is more robust in different environments and not overfit on the simulation environment in which it has been trained.

The randomisation is implemented in the Gazebo simulation using the Gazebo Awesome Plugins (GAP) [41]. GAP provides the additional functionality to randomise the Gazebo environment and is implemented as a ROS node.

The simulation environment is randomised at the start of each new flight according to Table 4.1. These distributions are based on the work from OpenAI *et al.* [31] as a guideline. The product of inertia elements of the multirotor are assumed zero during the controller design, but in reality this is not the case because the multirotor is not perfectly symmetrical. The upper bound was selected finding the lowest principle of inertia value of the multirotor and reducing it by an order of magnitude. The gravity vector is added with a Gaussian distribution, where the addition in the $\bar{\boldsymbol{x}}_{\mathcal{I}}$ and $\bar{\boldsymbol{y}}_{\mathcal{I}}$ direction is simulated as additional uncertainty in the dynamics of the multirotor.

4.1.2 Generation of Trajectories Setpoints

Generating a large dataset for training neural networks leans itself to the automation of generating and sending the setpoints to the flight controller. This was accomplished by using a ROS node, sending the generated trajectory to the flight controller during each simulated flight. The ROS node would also command the multirotor to take-off and land.

The ROS node sends velocity or position setpoints to the multirotor to execute, and the decision is based on the Bernoulli trial with a probability 0.5. The reason for sending either position or velocity setpoints is to allow the neural network to train on a flight envelope which is wide enough. If only velocity

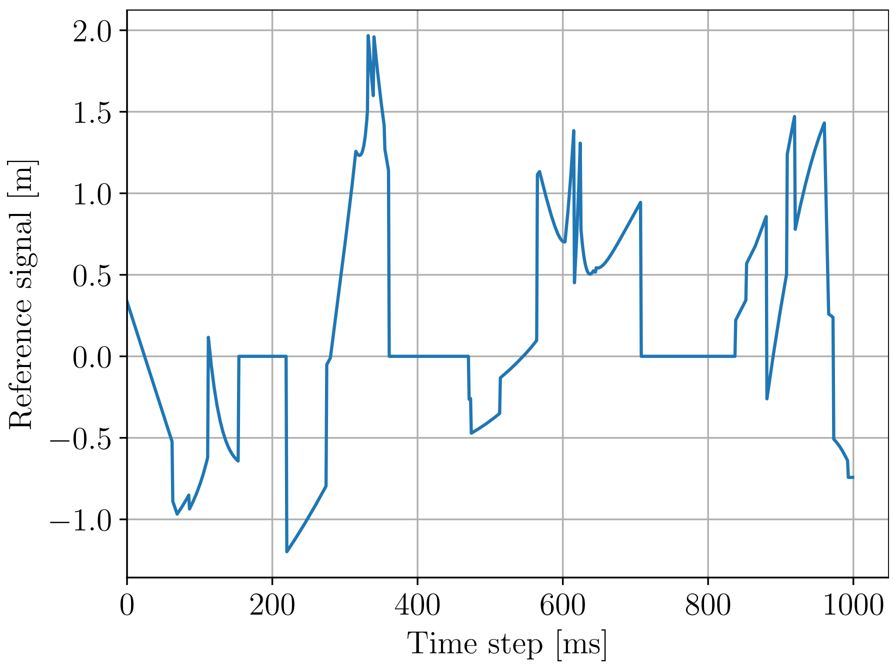


Figure 4.1: A random setpoint generated by Equation 4.1 containing the expected properties produced by waypoint flying and manual control.

setpoints were used, the neural network would never identify how a multirotor hovers. Furthermore, angle and angular rate setpoints were not used because it would result in the multirotor performing trajectories outside of the design region: performing acrobatic manoeuvres such as flips.

The velocity and position setpoints should contain the expected properties being produced by the controller, or a human and these were determined as step, ramp and exponential functions. When observing waypoint flying, the setpoint that the position controller receives is a step functions representing the new location. The velocity loop receives ramp inputs when executing waypoint flying because as it is flying towards the next waypoint, the positional error decreases with a given slope for any gain in Equation 3.2. The exponential and square functions were added as the additional waveforms required to approximate human input through a remote control.

The generated trajectory is governed by the following equation,

$$u(t) = \sum_{\mathrm{q}=1}^{4} Y_{\mathrm{q}} \cdot h_{\mathrm{q}}(t, \tau_{\mathrm{q}}), \tag{4.1}$$

where Y_{j} is a random magnitude selected from a uniform distribution and h_j is functions of the form shown in Table 4.2. Each variable describing their function is sampled from an uniform distribution over an appropriate interval. Fig. (4.1) shows the generated setpoint which contains the expected properties.

Function	Equation	Description	Random variable
h_1	$1(t-\tau_1)-1(t-\tau_1-\tau_2)$	Step	τ_1, τ_2
h_2	$e^{\tau t}$	Exponential	τ
h_3	$m(t-\tau)$	Ramp	m, τ
h_4	at^2+bt+c	Square	a, b, c

Table 4.2: Functions used to represent a setpoint produced by a linear controller or human.

The reason for characterising the setpoints using random variables is to protect the neural network from learning a correlation between the setpoints and the disturbances. If this was the case, the neural network would overfit on the training data and will not learn the underlying dynamics between the disturbance and the response of the multirotor.

4.1.3 Generation of Disturbances

A Gazebo plugin was created to apply forces and moments to the multirotor during the time of executing the random trajectories. The forces and moments are applied in the body frame of the multirotor and are described by a random

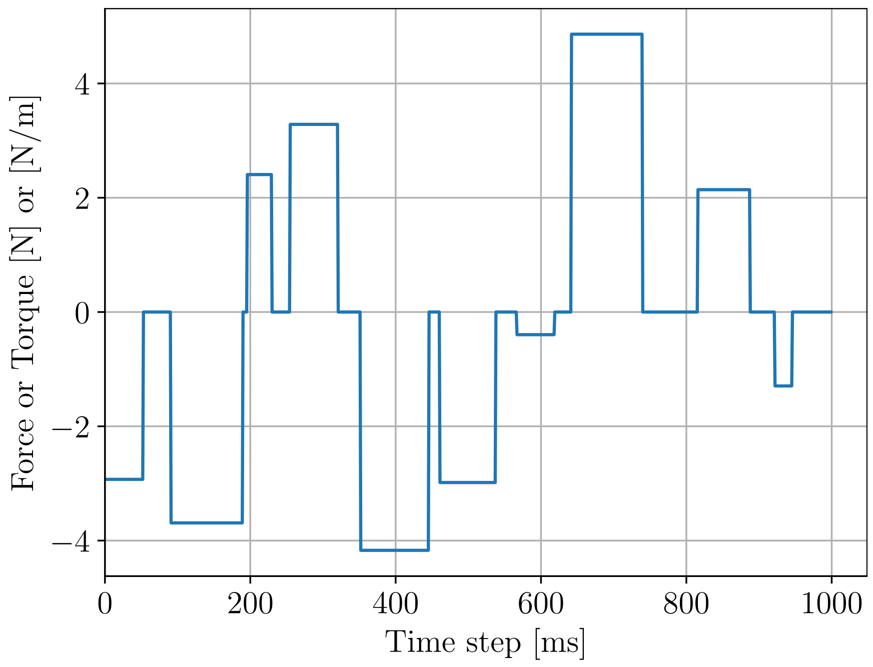


Figure 4.2: A random pulse train used as the disturbance effecting a multirotor.

pulse train and shown in Fig. (4.2). The neural network is required to identify any form of disturbance influencing the multirotor from its expected behaviour, and thus the disturbance must maintain an autocorrelation of zero throughout. If the disturbance has a non-zero autocorrelation, the neural network would learn this correlation and then not generalise the underlying dynamics of the response of the multirotor experiencing a disturbance. Previous work has shown that using random pulse trains as disturbances affecting the multirotor in simulation results in the trained neural network to generalise to unseen disturbances such as step and sinusoidal disturbances [8]. The maximum and minimum of the random pulse trains were limited to -2N and 2N to ensure a stable flight and the maximum pulse length limited to 8 seconds. During flight, the multirotor will be disturbed in each of the body directions with a unique force and torque random pulse trains. The disturbance signal is discretised at the same frequency at which the Gazebo simulation executes, which allows for easy logging and dataset creation.

4.2 Simulated Flight

A simulated flight starts by spawning the multirotor in the Gazebo environment and starting the PX4 flight controller, as shown in Fig. (4.3). Once the multirotor has spawned, the ROS node *Domain Randomisation* randomises the parameters of interest. From there the *Trajectory Generator* ROS node sends the take-off command, following the randomised trajectory. During the execution of the randomised trajectory, the Gazebo disturbance plugin is activated to disturb the multirotor as it is trying to execute the given trajectory. Once the disturbance has finished, the *Trajectory Generator* ROS node signals

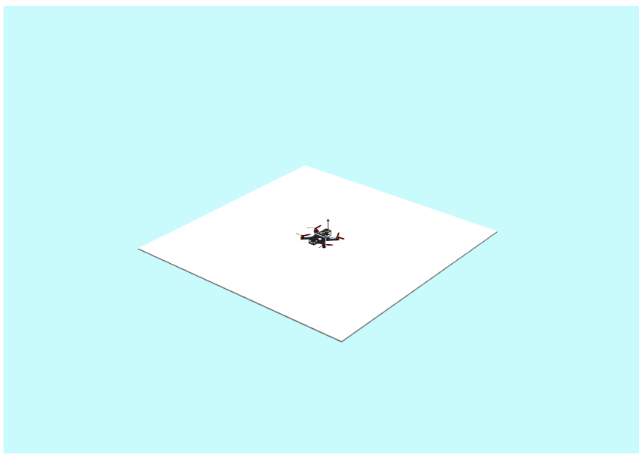


Figure 4.3: Quadcopter being spawned in Gazebo just before take-off command is given.

the multirotor to land where the *Domain Randomisation* ROS node generates the landing pad for the multirotor.

Fig. (4.4) shows a multirotor commanded to hover in place during which it is experiencing force disturbances in the $\bar{\boldsymbol{x}}_{\mathcal{B}}$-axis. To counteract the disturbance, the control systems commands the vehicle to angle the thrust produced by the motors against the direction of the disturbance. During the response, the position setpoint stayed constant, and it was expected that the multirotor would remain at steady state if no disturbances occurred. It is the difference between the expected behaviour, and the actual behaviour that results in the identification of disturbances and Fig. (4.4) shows the correlation between the disturbance and the multirotor response which the neural network must learn.

4.3 Dataset Creation

PX4 starts to log the various parameters of interest when the flight controller receives the arm command and stops the recording when a landing has been detected. The logged data of interest is summarised in Table 4.3 and is used to estimate the disturbance effecting the multirotor.

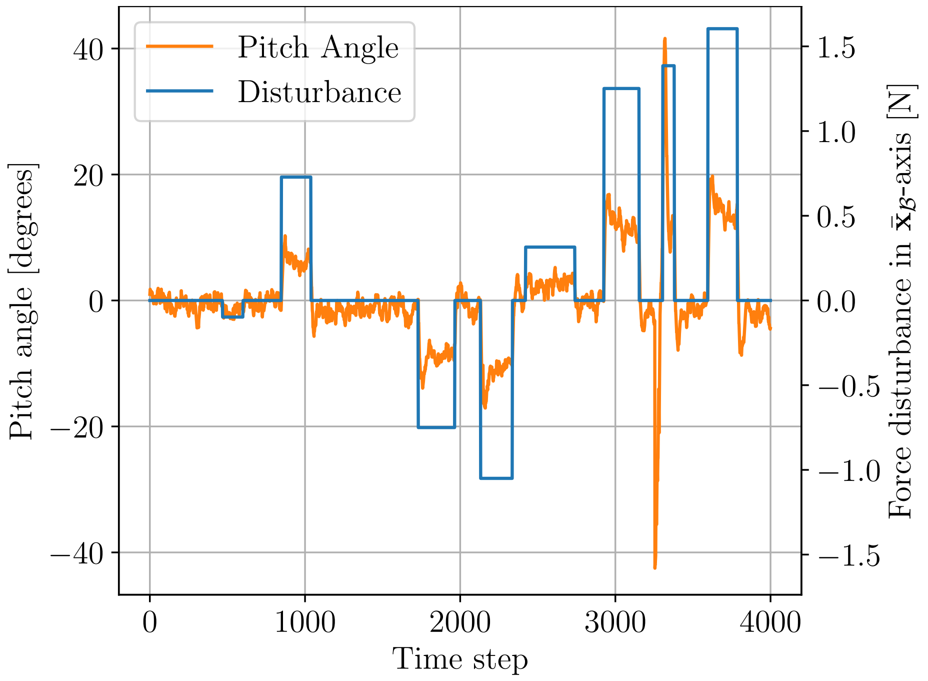


Figure 4.4: Pitch angle response of the multirotor under the influence of disturbances only in the $\bar{\boldsymbol{x}}_{\mathcal{B}}$-axis.

Parameter	Number of elements	Storing frequency
Inertial velocity	3	25
Inertial velocity setpoint	3	25
Quaternions	4	100
Quaternion setpoint	4	100
Angular Rates	3	250
Angular Rate setpoints	3	250
PWM signal to motors	4	250
Total	24	-

Table 4.3: Parameters of interest being stored during a simulated flight.

From Table 4.3, it is visible that each of the input and output pairs for the various controllers was selected excluding the position controller. It is important to provide these input and outputs pairs because the neural network is required to learn how the multirotor aught to behave given the reference. Any deviation from the expected behaviour is the manifestation of disturbances. The various inputs are also not stored at the same frequency, and thus the inputs at slower frequency are kept constant until it updates.

The dataset is standarised using,

$$Y' = \frac{Y - \mu}{s} \tag{4.2}$$

where μ and s refers to the mean and standard deviation respectively of the corresponding vector in the dataset. The standardisation of the dataset was based on [31] using the same preprocessing of the data.

4.4 Neural Network Architecture

The neural network used to estimate the disturbances affecting a multirotor is based on the work done by [31]. They combined the fully connected dense ReLU layer with a layer containing multiple LSTM units which then outputs the corresponding action the robot hand should execute. The project requires the estimated disturbance, and thus after the LSTM layer, a single layer with linear activation functions are used, which represents the disturbance in each direction and is shown in Fig. (4.5). β, γ refers to the number of units of the respective type used in the layer. [31] experimented with different architectures and suggested that initial interaction with the environment would reveal the global constants of the environment, such as the weight of the multirotor. [31] also experimented with other architectures and concluded that the LSTM is superior with regards to Sim2Real transfer.

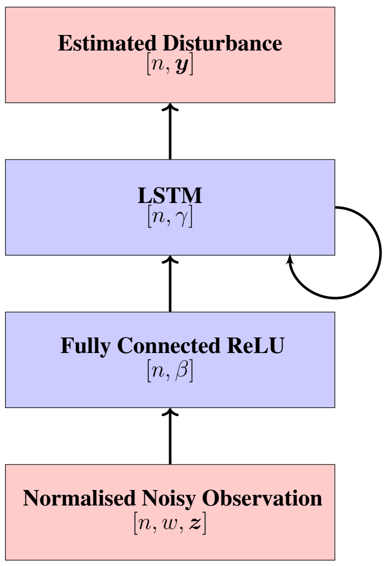


Figure 4.5: Neural network architectures used for learning disturbances.

The loss function used for training of the neural network is the mean absolute error (MAE) with weight regularisation,

$$L = \frac{1}{n}\sum_{i=0}^{n} \mid \boldsymbol{\Theta}(\boldsymbol{x}) - \boldsymbol{y} \mid + \lambda \sum_{l \in L}\sum_{j \in J}\sum_{k \in K} \mid W_{jk}^{(l)} \mid \tag{4.3}$$

and is used to associate the same importance between large and smaller errors. The optimiser used to update the trainable variables is Adam [42].

The input data to the neural network model is a vector of the form

$$\boldsymbol{x} = [\text{batchsize, windowsize, input vectors}] := [n, N, \boldsymbol{z}], \tag{4.4}$$

where the input vector has a size of 24 and correspond to the parameters shown in Table 4.3. The window size is the number of consecutive previous logged timestamps for each input vector provided, and the batchsize correspond to the number of corresponding groups of window size and input vectors. The input to the neural network thus has a size of $n \cdot N \cdot \text{sizeof}(\boldsymbol{z})$ and undergoes the nonlinear transformation to output the estimated disturbance in the form of

$$\boldsymbol{\Theta}(\boldsymbol{x}) = [\text{batchsize, output vectors}] := [n, \hat{\boldsymbol{y}}], \tag{4.5}$$

where $\hat{\boldsymbol{y}}$ is the estimated disturbance influencing the multirotor. During the nonlinear transformation, dropout is applied to the first fully-connected layer to improve the generalisation of the neural network.

4.5 Summary

Simulated data is used to train the neural network, which will be operating in reality, to estimate disturbances. This process is known as Sim2Real transfer. This chapter thus describes how the data generation is done for successfully training a neural network in simulation to operate in reality. This data generation refers to how the simulation environment is adapted, the generation of the reference signal that the multirotor should execute, how the disturbances are modelled and then how the dataset is generated. All of the previously mentioned points focus on guarding the neural network against overfitting the simulation environment and learning the correlation between the response of the multirotor being influenced by a disturbance.

Chapter 5

Estimation Results of Observers

This chapter will introduce the classical techniques used for estimating disturbances effecting a multirotor and provide results of their estimation ability for various types of disturbances. These classical disturbance techniques are used to provide a comparison against the performance of the neural network. It follows with the neural network's estimation of disturbances effecting a multirotor from the validation dataset as well as unseen disturbances to confirm the generalisation ability of the neural network. For each observer, commentary is provided regarding their performance on estimating disturbances. Lastly, the process of modelling the neural network in MATLAB is described to compare the disturbance rejection ability to the other observers.

5.1 Extended Kalman Filter

The EKF is an estimator of nonlinear systems which takes inputs that are assumed to be Gaussian distributed. It is a recursive algorithm containing two steps: the model and the measurement update, as shown in Fig. (5.1). It is used extensively as an estimator in robotic systems. Referring to Fig. (5.1), $\boldsymbol{F}_k$ and $\boldsymbol{H}_k$ are the discrete Jacobian matrix of $\boldsymbol{f}(\cdot)$ and $\boldsymbol{h}(\cdot)$ which are the nonlinear process and measurement model, respectively. The Jacobian matrices are computed using the previous timestep estimate of the states and covariance matrix. Finally, $\boldsymbol{R}_k$ and $\boldsymbol{Q}_k$ are the process and measurement noise matrices and $\boldsymbol{L}_k$ the Kalman gain, which is an optimally weighting matrix to update the states based on the uncertainty between the measurement and the states predicted by the model. For an in-depth mathematical derivation and understanding from a control perspective, refer to [43].

The EKF is not limited to the multirotor states and can estimate the disturbances of the environment affecting the multirotor. This was practically demonstrated by [21] who added a disturbance model to the EKF to estimate

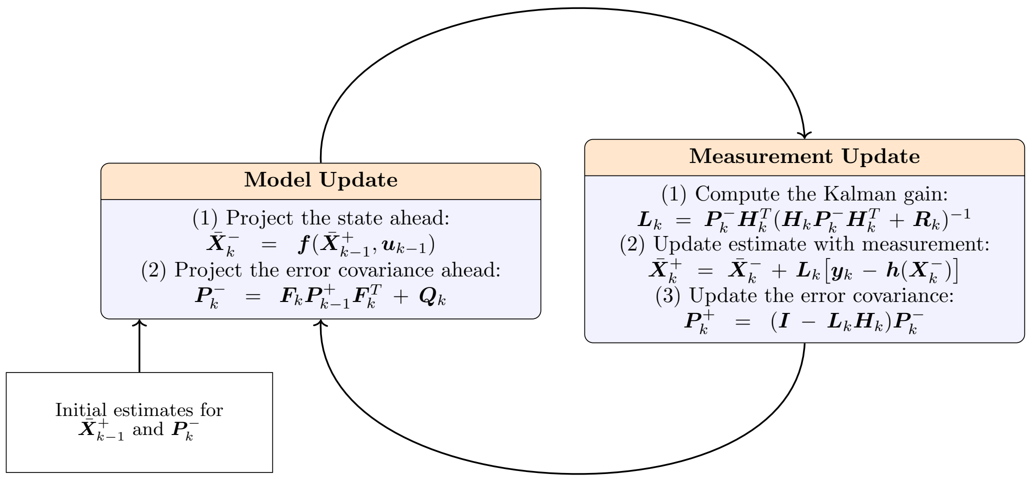

Model Update
(1) Project the state ahead:
$\bar{\boldsymbol{X}}_k^- = \boldsymbol{f}(\bar{\boldsymbol{X}}_{k-1}^+, \boldsymbol{u}_{k-1})$
(2) Project the error covariance ahead:
$\boldsymbol{P}_k^- = \boldsymbol{F}_k \boldsymbol{P}_{k-1}^+ \boldsymbol{F}_k^T + \boldsymbol{Q}_k$
Measurement Update
(1) Compute the Kalman gain:
$\boldsymbol{L}_k = \boldsymbol{P}_k^- \boldsymbol{H}_k^T (\boldsymbol{H}_k \boldsymbol{P}_k^- \boldsymbol{H}_k^T + \boldsymbol{R}_k)^{-1}$
(2) Update estimate with measurement:
$\bar{\boldsymbol{X}}_k^+ = \bar{\boldsymbol{X}}_k^- + \boldsymbol{L}_k[\boldsymbol{y}_k - \boldsymbol{h}(\boldsymbol{X}_k^-)]$
(3) Update the error covariance:
$\boldsymbol{P}_k^+ = (\boldsymbol{I} - \boldsymbol{L}_k \boldsymbol{H}_k) \boldsymbol{P}_k^-$
Initial estimates for $\boldsymbol{X}_{k-1}^+$ and $\boldsymbol{P}_k^-$

Figure 5.1: The EKF recursive algorithm used for estimating the disturbances affecting the multirotor.

force disturbances acting upon a multirotor. They modelled the disturbance as a random walk process described by:

$$\boldsymbol{y}_t = \boldsymbol{y}_{t-1} + \boldsymbol{d}_t \tag{5.1}$$

where $\boldsymbol{d}_t$ is assumed to be Gaussian white noise such that $\boldsymbol{d}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{s^2})$. This model of disturbances will be used in the model-based EKF, and the converging speed of the EKF to the estimated disturbance is tuned by adjusting the variance of the random walk: increasing the variance results in a faster convergence speed at the cost of a noisier estimate. The mathematical equation governing the multirotor kinematics in Equation 2.1 is expanded to include the force and moment disturbances and results in,

$$\begin{aligned} \boldsymbol{F}_\mathcal{B} &= \boldsymbol{F}_\mathcal{B}^T + \boldsymbol{F}_\mathcal{B}^G + \boldsymbol{F}_\mathcal{B}^D \\ \boldsymbol{M}_\mathcal{B} &= \boldsymbol{M}_\mathcal{B}^T + \boldsymbol{M}_\mathcal{B}^G + \boldsymbol{M}_\mathcal{B}^D \end{aligned} \tag{5.2}$$

with the superscript D referring to the disturbances.

5.1.1 MATLAB Implementation

The EKF is implemented in a MATLAB environment, alongside the identical PX4 controllers used in the SITL simulation, which was presented in Chapter 3. The model-based EKF would receive its measurements from the PX4 EKF estimates, and for this reason, the effect of sensor drift is assumed negligible and omitted. In the practical experiment of [21], the model-based EKF was

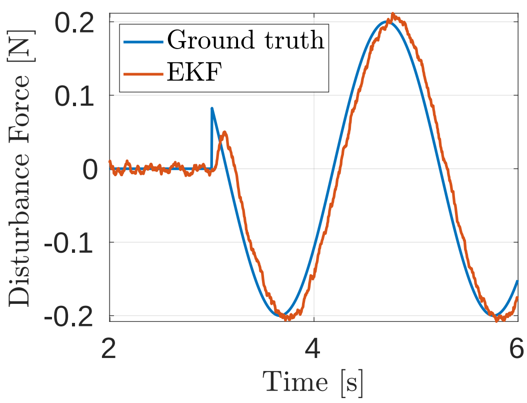


Figure 5.2: EKF estimating a sinusoidal force disturbance in the $\bar{\boldsymbol{x}}_{\mathcal{B}}$ direction of the multirotor.

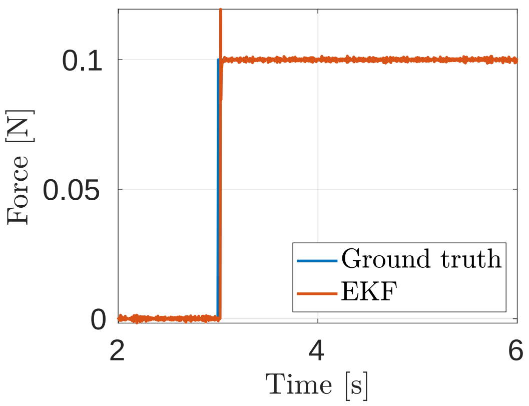


Figure 5.3: EKF estimating a step torque disturbance in the $\bar{\boldsymbol{y}}_{\mathcal{B}}$ direction of the multirotor.

executing on a "offboard" computer [44] which receives the estimated states across a communication layer and thus a communication delay of 50Hz is included. Fig. (5.4) shows the architecture of this process and is similarly used in [21] for disturbance rejection. The MATLAB environment is further enhanced by sampling the mass and moment of inertia values from a Gaussian distribution. Its mean is the calculated values of the multirotor in Table 3.2, and the variance of the respective random variable is shown in Table 5.1. This is done to simulate a more realistic performance from the EKF.

Fig. (5.3) shows the EKF estimating a torque disturbance simulated as a step function. The variance of the random walk was tuned to provide a satisfactory tradeoff between noise and bandwidth and simultaneously making the results comparable to the other techniques implemented.

The EKF is likewise capable of estimating time-varying disturbance, as shown

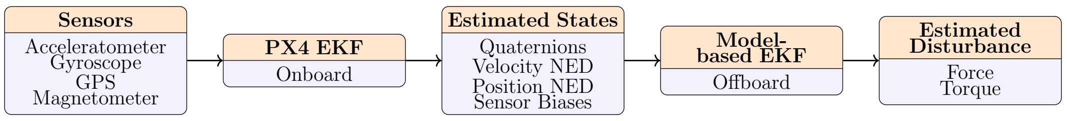


Figure 5.4: Process being emulated in MATLAB.

Parameter	Variance
Mass	0.01
Moment of inertia terms	0.00001

Table 5.1: Gaussian distribution used for the physical properties of the multirotor.

in Fig. (5.2), which is simulated as a force disturbance. These results are exclusively estimating the disturbance affecting the multirotor without it being rejected. The introduction of the disturbance rejection will influence the EKF's estimation of the disturbance since any overshoot or error will introduce a proportional disturbance to the error. From these results, it is evident that the model-based EKF is capable of estimating the disturbance affecting the multirotor.

5.2 Extended State Observer

Estimators of disturbances are generally the Extended State Observers (ESO) when used in a disturbance rejection architecture in a variety of fields [45], [46], [47]. They have also been used to estimate disturbances affecting a multirotor. The ESO is a linear current estimator which operates on Single-Input Single-Output (SISO) systems and live in the state space domain, as shown in Fig. (5.5). The ESO yields the matrix $\boldsymbol{L}$, shown in Fig. (5.5), which causes the estimates to converge on the actual values. Construction of the ESO starts by the differential equation,

$$y^n(t) = f(y(t), \dot{y}(t), ..., y^{n-1}(t), d(t), t) + bu(t) \tag{5.3}$$

where $d(t)$ is the external disturbances, $y(t)$ the controlled output, $u(t)$ the control input and b a system gain. Equation (5.3) can be rewritten to be represented as

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = x_3 \\ \vdots \\ \dot{x}_{n-1} = x_n \\ \dot{x}_n = f(x_1, x_2, \ldots, x_n, d(t), t) + bu \\ y = x_1. \end{cases} \tag{5.4}$$

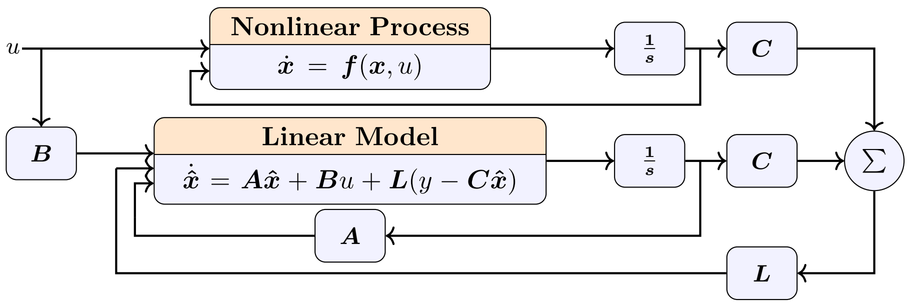

u
Nonlinear Process
$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, u)$
$\frac{1}{s}$
$\boldsymbol{C}$
$\boldsymbol{B}$
Linear Model
$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{A}\hat{\boldsymbol{x}} + \boldsymbol{B}u + \boldsymbol{L}(y - \boldsymbol{C}\hat{\boldsymbol{x}})$
$\frac{1}{s}$
$\boldsymbol{C}$
Σ
$\boldsymbol{A}$
$\boldsymbol{L}$

Figure 5.5: State space representation of a linear estimator.

In the framework of the ESO, an additional state is included which represents the lumped disturbances in the system,

$$x_{n+1} = f(x_1, x_2, \dots, x_n, d(t), t), \tag{5.5}$$

and by combining Equation 5.5 and Equation 5.4 results in

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = x_3 \\ \vdots \\ \dot{x}_{n-1} = x_n \\ \dot{x}_n = x_{n+1} + bu \\ \dot{x}_{n+1} = \dot{f}(x_1, x_2, \dots, x_n, d(t), t) \\ y = x_1. \end{cases} \tag{5.6}$$

The ESO is then designed to estimate the states of the system where x_{n+1} represent the lumped disturbances.

5.2.1 MATLAB Implementation

Similar to the EKF, the ESO is implemented in MATLAB alongside the PX4 controllers. The discrete equivalent ESO's are designed for the angular rate and velocity SISO plants of the multirotor for estimating torque and force disturbances respectively. The mass and moment of inertia elements are also sampled from a Gaussian distribution using the variance from Table 5.1.

Fig. (5.6) shows the ESO estimating a sinusoidal torque disturbance affecting the multirotor in the $\bar{\boldsymbol{y}}_{\mathcal{B}}$-axis. It can be seen that the ESO is capable of

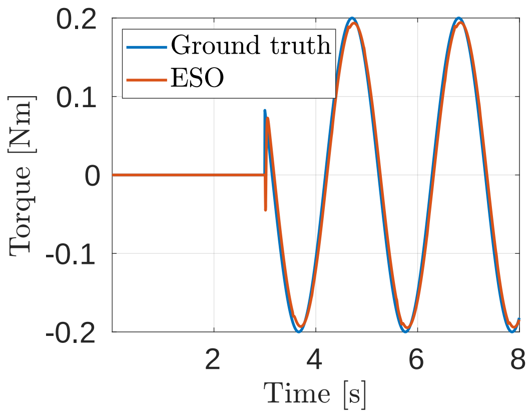


Figure 5.6: ESO estimating sinusiodale torque disturbance affecting the multirotor in the $\bar{\boldsymbol{y}}_{\mathcal{B}}$ direction.

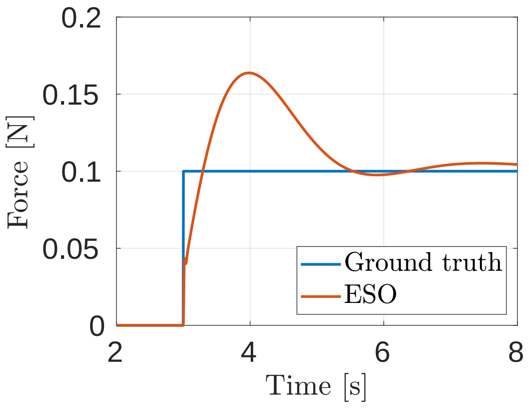


Figure 5.7: ESO estimating a step force disturbance affecting the multirotor in the $\bar{\boldsymbol{x}}_{\mathcal{B}}$-axis.

Hyperparameters	Value
Hardware configuraion	Nvidia Titan Xp
Optimiser	Adam [42]
Learning rate (α)	$3 \cdot 10^{-4}$
Weight regularisation (λ)	$7 \cdot 10^{-4}$
Batchsize (n)	128
Dropout ratio (p)	0.4
LSTM size (γ)	64
ReLU layer size (β)	128
Time window size (N)	300

Table 5.2: Hyperparameter values used for training.

estimating the disturbance rather quickly. This quick response is due to the low model uncertainty in the linear angular rate model. Fig. (5.7) shows the ESO estimating a step force in the $\bar{\boldsymbol{x}}_{\mathcal{B}}$-direction, and exhibits rather poor characteristics such as large overshoot and a long settling time. This response is attributed to the fact that the velocity plant contains more uncertainty in the plant dynamics as the angular rate plant.

5.3 Neural Network

The neural network is trained on a dataset containing 775 independent flight logs with an average flight time of 80 seconds which includes landing and takeoff. The training dataset was generated by randomly selecting 90% of the 775 independent flight logs and stitching them together. The remaining 10% is used for validation to examine whether overfitting has occurred. The tuning of the hyperparameters occurred using only 5% of the training dataset and was done for faster identification of the appropriate ranges to use on the full dataset. The final values for the hyperparameters are shown in Table 5.2.

The training[1] occurred for 168 hours and resulted in the MAE of 0.13. The loss value over this period is shown in Fig. (5.8), and it is visible that the loss value has not converged yet due to interruption by the HPC scheduler. The use of the LSTM in the neural network dramatically increases the training time but is crucial for the Sim2Real transfer. It is suggested by [31] that the LSTM is capable of identifying the environmental parameters from the time window and results in improved generalising of the neural network. Two other architectures were trained to examine whether they provide improvements over the OpenAI architecture. These two architectures replaced the LSTM layer

[1]Computations were performed using the University of Stellenbosch's HPC1 (Rhasatsha): http://www.sun.ac.za/hpc

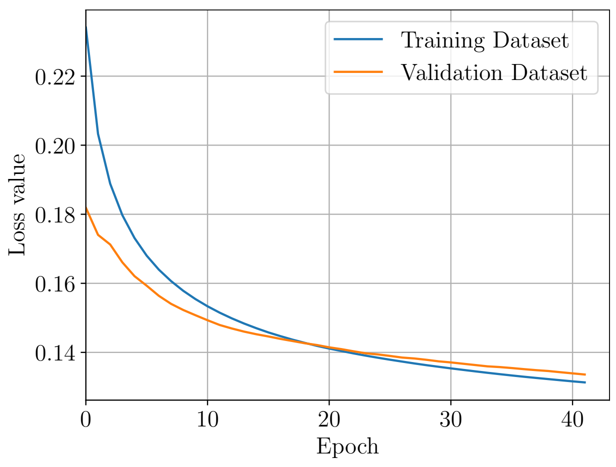


Figure 5.8: Loss value during training of the neural network.

with either a fully-connected layer using ReLU activation functions or with an alternative recurrent layer known as the Gated Recurrent Unit (GRU). Their performance is summarised in Table 5.3, but neither architectures are able to reach similar accuracy as the LSTM, though they do train significantly faster. These results confirm the findings of [31].

Even though the neural network has converged to an acceptable MAE value on the training dataset, it must perform to a similar degree on data it has not been trained on and tested on other types of disturbances. Fig. (5.9), (5.10) and Fig. (5.11) visually verifies that the neural network has not overfitted by estimating the disturbances contained in the validation dataset. It is visible that the neural network is able to estimate the disturbances; however, there are regions where the neural network overestimate the disturbance. There are also cases in the validation dataset which the neural network falsely identifies a disturbance when, in fact, there is none. This behaviour is a direct consequence of the neural network underfitting the training data, and with further training, the neural network is expected to perform better. It should be noted that during this estimation, the multirotor is experiencing disturbances from all three directions, which is unrealistic but will improve the ability of the neural network to estimate disturbances in reality.

The generalisation of the neural network is established by estimating disturbances of a different nature. Fig. (5.13) shows the neural network estimating a time-varying disturbance only present in the $\bar{\boldsymbol{x}}_{\mathcal{B}}$-direction and Fig. (5.12) shows the neural network estimating a step disturbance. From these results, the neural network shows to have learned the underlying dynamics to a certain degree. Though the neural network does not perfectly estimate these unseen disturbances, it shows promising results for an under fitted neural network, and as mentioned, with further training, improved results are expected.

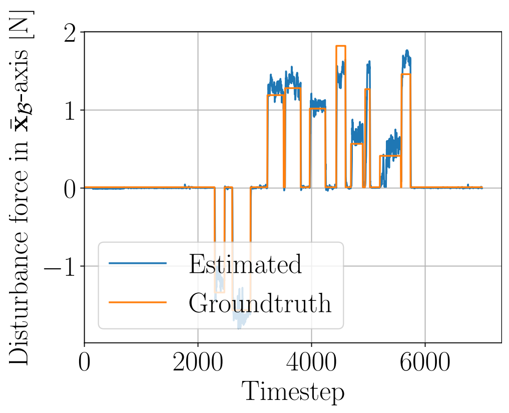


Figure 5.9: Neural network estimating the disturbance force affecting the multirotor in the $\bar{\boldsymbol{x}}_{\mathcal{B}}$-direction while being disturbed in the $\bar{\boldsymbol{y}}_{\mathcal{B}}$ and $\bar{\boldsymbol{z}}_{\mathcal{B}}$ direction shown in Fig. (5.10) and Fig. (5.11).

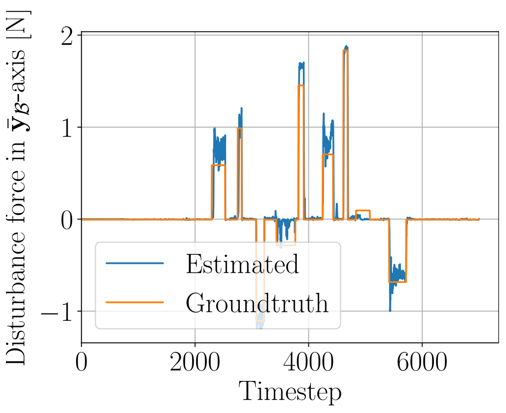


Figure 5.10: Neural network estimating the disturbance force affecting the multirotor in the $\bar{\boldsymbol{y}}_{\mathcal{B}}$ direction while being disturbed in the $\bar{\boldsymbol{x}}_{\mathcal{B}}$, and $\bar{\boldsymbol{z}}_{\mathcal{B}}$ direction shown in Fig. (5.9) and Fig. (5.11).

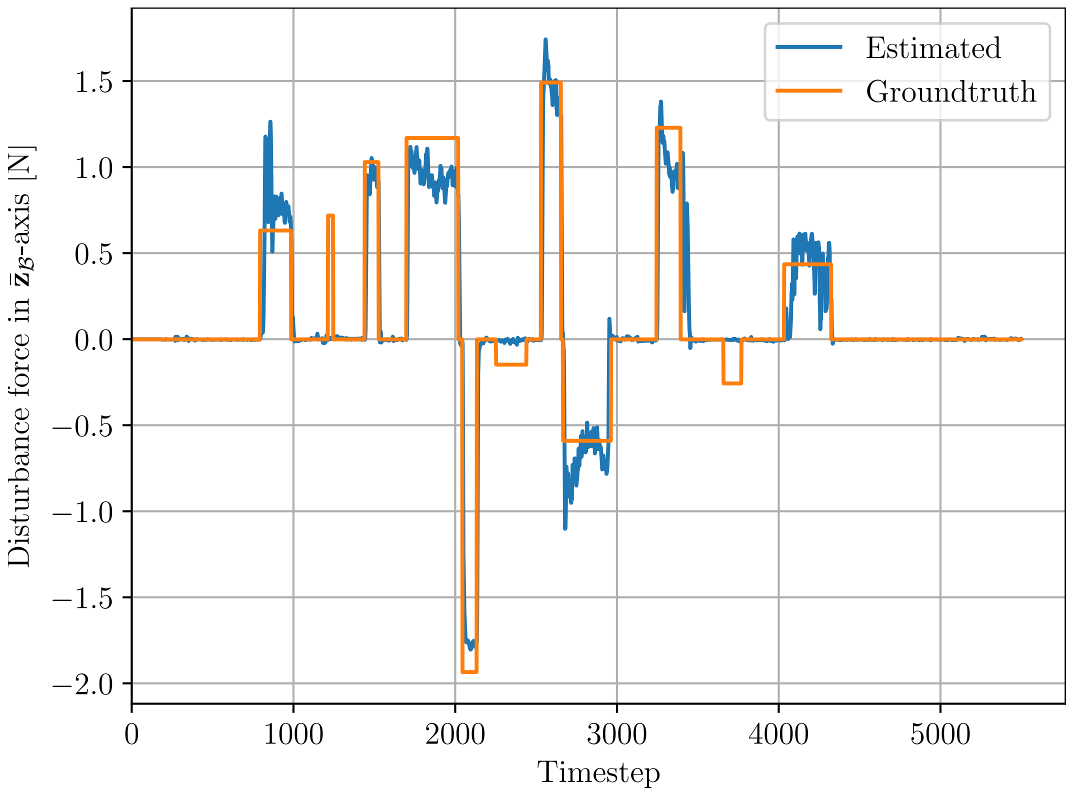


Figure 5.11: Neural network estimating the disturbance force affecting the multirotor in the $\bar{\boldsymbol{z}}_{\mathcal{B}}$ direction while being disturbed in the $\bar{\boldsymbol{x}}_{\mathcal{B}}$ and $\bar{\boldsymbol{y}}_{\mathcal{B}}$ direction shown in Fig. (5.9) and Fig. (5.10).

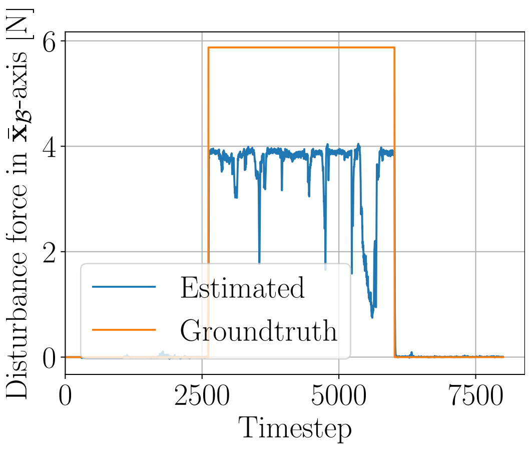


Figure 5.12: Neural network estimating a step force disturbance affecting the multirotor in the $\bar{\boldsymbol{x}}_{\mathcal{B}}$-direction.

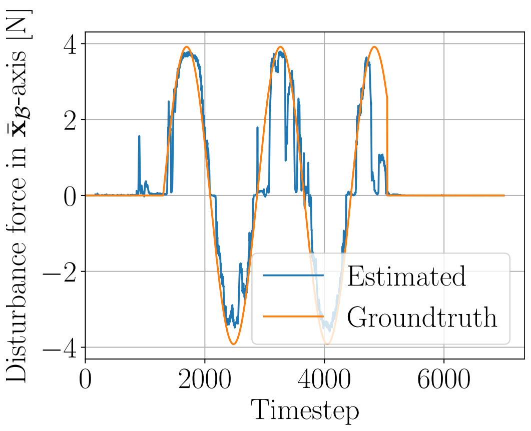


Figure 5.13: Neural network estimating a sinusoidal force disturbance affecting the multirotor in the $\bar{\boldsymbol{x}}_{\mathcal{B}}$-direction.

5.3.1 Estimation during Practical Flight Test

Fig. (5.14) shows the estimates of the neural network on a practical flight for the multirotor, *Honeybee* and Fig. (5.15) shows the position estimates during which the multirotor takes-off and then fly to a specific altitude. From these results, the neural network seems to transfer well to reality since no disturbance is estimated throughout the flight. The neural network uses a time window of 300 timesteps and thus does not predict anything initially during this time window. During the flight, the flight controller mode was changed to a mode which is not contained in the simulation resulting in the neural network to output zero near the end of Fig. (5.14). This flight mode causes certain setpoints to be set to not-a-number (NaN), and thus the neural network would not have previously seen this combination of setpoints.

To ensure that the previously shown practical flight test result was not a coincidence, Fig. (5.16) shows the estimation of the neural network from a different practical flight test with the estimated position of the multirotor in Fig. (5.17). During this flight test, the multirotor carried an additional payload and would require more thrust to stay in the air. This additional thrust

Model (number of units)	Validation MAE	Training time (hours)
Dense ReLU + LSTM(64)	0.13	168
Dense ReLU + GRU(64)	0.2	48
Dense ReLU + Dense ReLU(128)	0.28	48

Table 5.3: Comparisons between different neural network architectures.

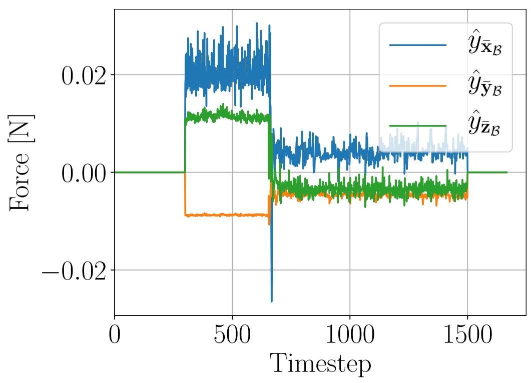


Figure 5.14: Neural network estimating the disturbances from a practical flight test.

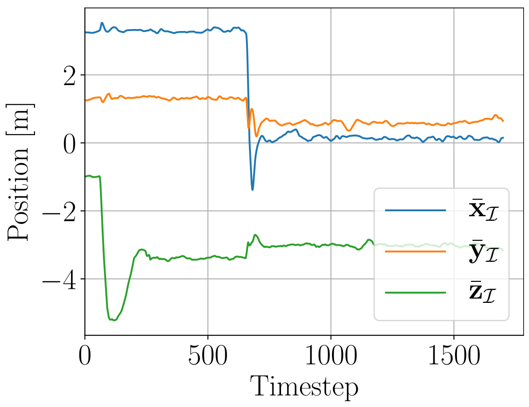


Figure 5.15: The position estimates of *Honeybee* from a practical test flight which correspond to Fig. (5.14).

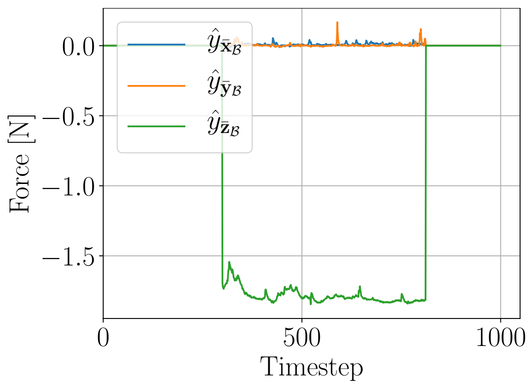


Figure 5.16: NN estimating disturbances from a practical flight test during which *Honeybee* carried a payload.

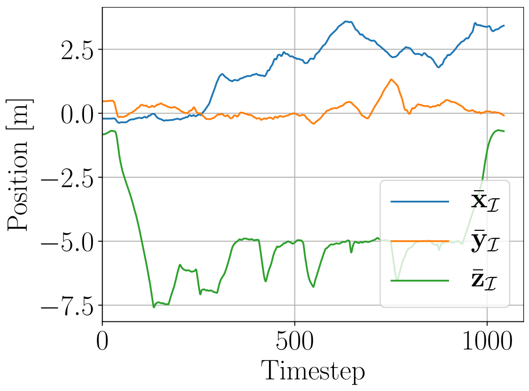


Figure 5.17: The position estimates of *Honeybee* from a practical test flight which correspond to Fig. (5.16).

translates to a disturbance influencing the multirotor in the $\bar{z}_I$ direction. It is visible from Fig. (5.16) that the neural network estimates a payload of 0.18kg in the $\bar{z}_I$ direction. During this practical flight test, the multirotor carried a payload of 0.14kg which practically verifies the estimation ability of the neural network. A similar practical flight test was conducted with the multirotor carrying again a 0.14kg payload shown in Fig. (5.19) with the estimation from the neural network shown in Fig. (5.18). The neural network estimates of the payload are satisfactory, and it is stable throughout the flight and shows that the methods used to improve the Sim2Real transfer were successful. With further training, these estimates are expected to improve since the trained neural network has underfitted the training dataset.

Fig. (5.20) again shows the neural network estimating a payload of 0.18kg

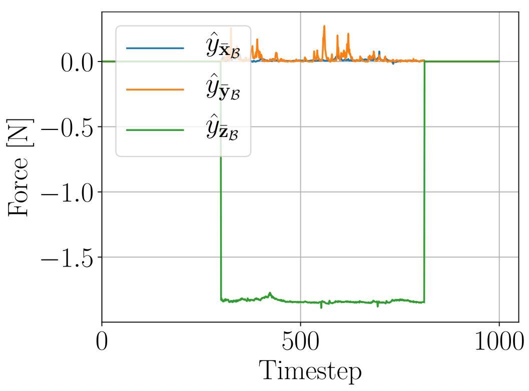


Figure 5.18: NN estimating disturbances from a practical flight test during which *Honeybee* carried a payload.

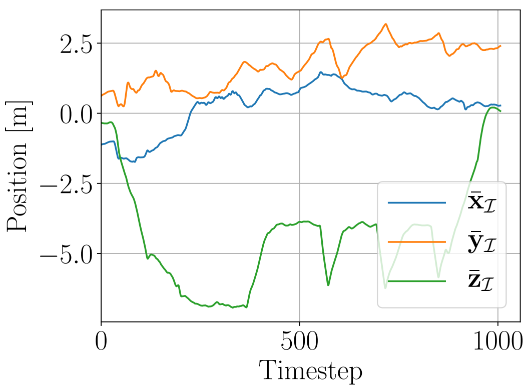


Figure 5.19: The position estimates of *Honeybee* from a practical test flight which correspond to Fig. (5.18).

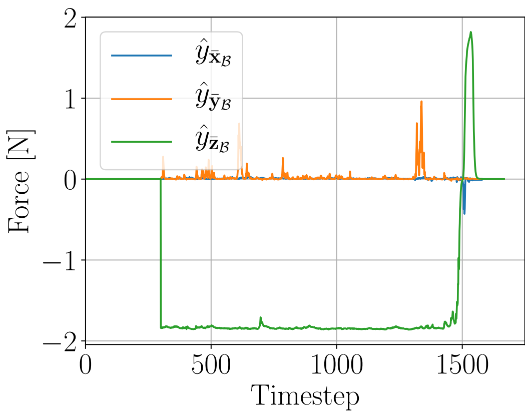


Figure 5.20: NN estimating disturbances from a practical flight test. The NN estimates a payload of 0.18kg and then ground effects during landing.

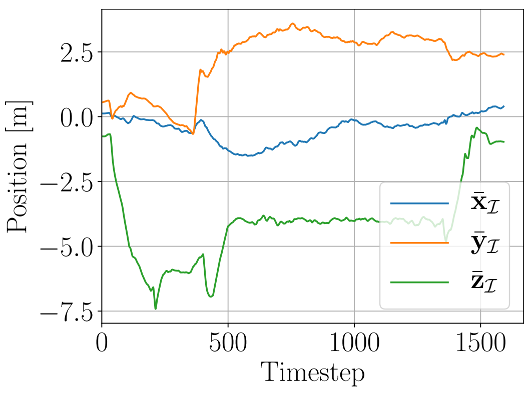


Figure 5.21: The position estimates of *Honeybee* from a practical test flight which correspond to the disturbances estimates of Fig. (5.20).

throughout the flight test with the position estimates in Fig. (5.21). During this flight test, the flight mode did not changed before landing and Fig. (5.20) shows the neural network estimating the ground effects at timestep 1500. During the practical flight tests, there were no measurements taken to estimate the ground effects and thus there are no ground truth to compare this result. However, it is promising that the neural network identifies disturbances during landing.

5.3.2 MATLAB Implementation

The neural network must be modelled in some way to be able to compare its disturbance rejection capabilities against the classical disturbance observers since it will be tested in a MATLAB environment. It was decided to model the neural network as three independent Gaussian random variables with each random variable being the disturbances in each respective direction. The mean of the random variables are the ground truth disturbance, and the variance was selected as a linear function based on the ground truth value of the disturbance and the MSE that the neural network achieved on the validation dataset. The MSE is the variance for an unbiased estimator as shown in Zheng [48] and was the best metric available to choose from to model the neural network as a Gaussian random variable. The neural network is thus modelled as,

$$\hat{\boldsymbol{y}}_t = \boldsymbol{y}_t + \boldsymbol{\mathcal{N}}(\boldsymbol{y}_t, \boldsymbol{y}_t \cdot 0.1), \tag{5.7}$$

with the gain of 0.1 in Equation 5.7 scaling the standard deviation from 0 to 0.4 across the total range of 4N which the neural network was trained on. Modelling the neural network as a Gaussian random variable ignores effects which could be expected during estimation, which are: large absolute errors from the ground truth values and sudden spikes in the estimation. It is also assumed that the introduction of rejection does not affect the estimation of the neural network, and this was observed in previous work [8]. The results presented should thus be seen as the best-case scenarios until the neural network is validated by using the actual trained model.

The results and discussion of the disturbance rejection performance of a neural network modelled by a Gaussian distribution is presented in the following chapter along the with the other observers.

5.4 Summary

This chapter presented the three observers which will be used in a disturbance rejection architecture which are the ESO, EKF and neural network. Each of their estimation of various disturbances was provided without the estimation being used in feedback because the introduction of feedback would affect their estimation. Both the ESO and EKF were able to estimate step and time-varying disturbance for both torque and force disturbances. The training of the neural network showed that the neural network has underfitted the training dataset, but is able to estimate unseen disturbances such as step and time-varying disturbances. It is expected with further training that these results will improve. The estimation of practical flight data shows that the neural network is capable of estimating ground effects and payloads. This confirms the Sim2Real transfer of the neural network. To test the neural network estimation in feedback, the method of modelling the neural network as a Gaussian random variable was explained. Next, the estimation of the disturbance will be introduced into feedback to provide rejection to the various disturbances.

Chapter 6

Disturbance Rejection Results

This chapter describes how the various estimators are incorporated into the PX4 control architecture in order to reject disturbances of various forms. The different estimators' disturbance rejection performance is then provided and discussed, which is followed by a comparison of the estimators. The comparisons make use of a quantitative method to score their disturbance rejection in a controlled environment. Commentary is provided in light of the estimators' scores.

6.1 Disturbance Rejection Architecture

The disturbance affecting the multirotor will be rejected by subtracting the estimated disturbance from the pseudo control signal generated by the PX4 controller, as shown in Fig. (6.1). Mathematically this can be expressed as

$$\acute{\boldsymbol{\delta}} = \boldsymbol{\delta}_{PX4} - \boldsymbol{K}\hat{\boldsymbol{y}}. \tag{6.1}$$

The PX4 controllers are designed using normalised gains, which implies that the controller outputs a value in the range of $[-1; 1]$. Refer to [32] for a derivation for how it is incorporated into the PX4 controllers. The estimated disturbance thus needs to be normalised, to ensure the disturbance is rejected.

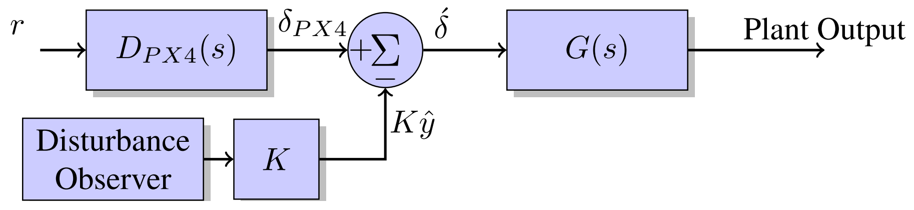
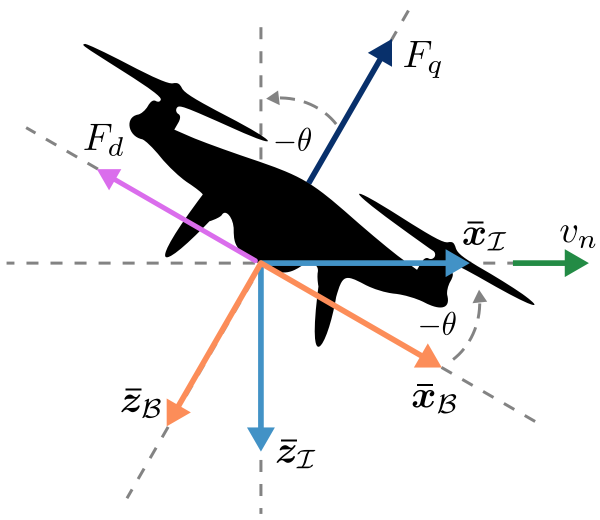


Figure 6.1: The control architecture used for angular rate and velocity subsystem to reject disturbances.



F_q
F_d
$-\theta$
$\bar{\boldsymbol{x}}_{\mathcal{I}}$
v_n
$-\theta$
$\bar{\boldsymbol{z}}_{\mathcal{B}}$
$\bar{\boldsymbol{x}}_{\mathcal{B}}$
$\bar{\boldsymbol{z}}_{\mathcal{I}}$

Figure 6.2: The free-body diagram of the multirotor flying at a constant longitudinal velocity and height.

This normalised gain is indicated by the K gain. The subtraction occurs in the subsystem which the disturbance manifests itself directly. This can be identified by examining the free-body diagram of the multirotor in Fig. (6.2) flying at a constant longitudinal velocity and height. In Fig. (6.2) Fq is the force produced by the propellers, F_d is the disturbance force in the $\bar{\boldsymbol{x}}_{\mathcal{B}}$ direction, m is the mass of the multirotor and g is the gravitational acceleration constant. Since the multirotor is flying at a constant velocity in the $\bar{\boldsymbol{x}}_{\mathcal{I}}$ direction the forces in all directions are in equilibrium. The sum of all the forces in the $\bar{\boldsymbol{z}}_{\mathcal{I}}$ direction results in

$$-F_q \cos(-\theta) - F_d \sin(-\theta) - mg\cos(\theta) = 0. \tag{6.2}$$

Assuming, for now, that there are no disturbances and applying the small angle approximation yields the following

$$F_q \approx mg. \tag{6.3}$$

The force in the $\bar{\boldsymbol{x}}_{\mathcal{I}}$ direction acting on the multirotor is,

$$F_{\mathcal{I}_N} = F_q \sin(-\theta), \tag{6.4}$$

and by substituting Equation 6.3 and assuming the small angle approximation results in,

$$F_{\mathcal{I}_N} = -mg\theta. \tag{6.5}$$

Remembering the relationship between Euler angles and quaternions discussed in Chapter 2, and by obtaining the longitudinal dynamics using Newton's second law which is,

$$F_{\mathcal{I}_N} = m\dot{V}_N = 2mgq_2, \tag{6.6}$$

the linear plant can be calculated using the Laplace Transform as:

$$G_{V_N}(s) = \frac{V_N(s)}{q_2(s)} = \frac{2g}{s}. \tag{6.7}$$

The velocity controller for the plant shown in Equation 6.7 will output a force which is translated into a reference angle for the angle controller. Thus, if a force disturbance is present, it will be rejected by subtracting the estimated disturbance force by the force generated by the velocity controller as shown

$$F_d - \hat{F}_d + F_c = m\dot{V}_N, \tag{6.8}$$

where $\hat{F}_d$ is the estimated disturbance force and F_c the force produced by the velocity controller. This is similarly identified in the sum of moment equation for the multirotor with a disturbance torque.

Thus, the estimated force disturbance will be subtracted in the velocity subsystem and the estimated torque disturbance in the angular rate subsystem. From a high-level perspective, Fig. (6.3) shows how the estimators for forces and torques are incorporated into the cascaded control architecture to reject the effect of the disturbances. Fig. (6.3) separates the force and torque observers however, this is not the case for the model-based EKF and neural network which combines them.

Next, the three observers' disturbance rejection performance is discussed. They are placed in the disturbance rejection architecture, with the estimated disturbance being used in feedback in the manner described above.

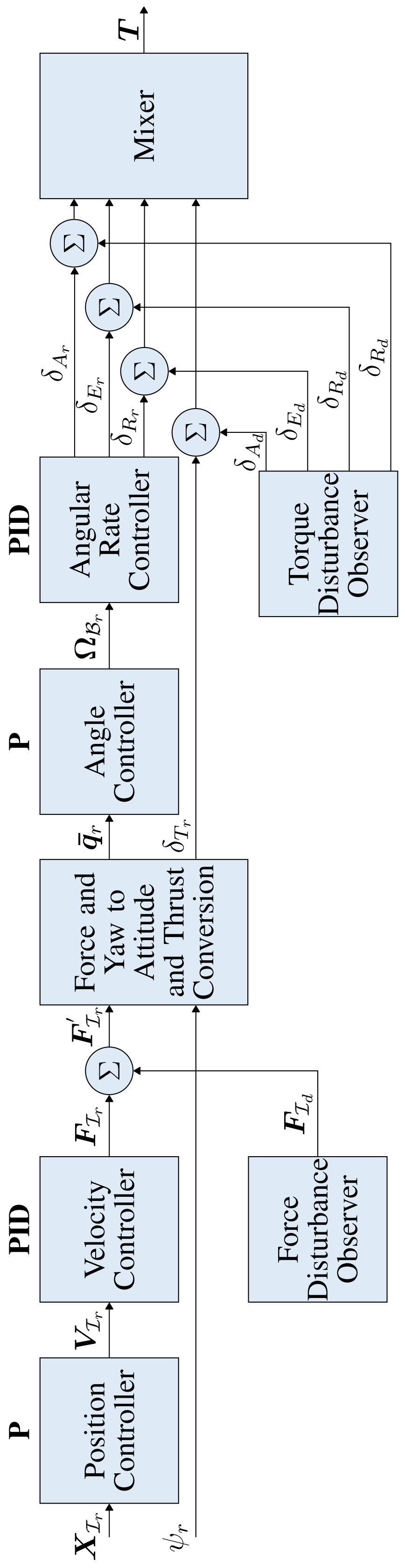

$\boldsymbol{X}_{\mathcal{I}_r}$
ψ_r
P
Position Controller
$\boldsymbol{V}_{\mathcal{I}_r}$
PID
Velocity Controller
Force Disturbance Observer
$\boldsymbol{F}_{\mathcal{I}_r}$
$\boldsymbol{F}_{\mathcal{I}_d}$
Σ
$\boldsymbol{F}'_{\mathcal{I}_r}$
Force and Yaw to Attitude and Thrust Conversion
$\bar{\boldsymbol{q}}_r$
δ_{T_r}
P
Angle Controller
$\boldsymbol{\Omega}_{\mathcal{B}_r}$
PID
Angular Rate Controller
Torque Disturbance Observer
δ_{A_r}
δ_{E_r}
δ_{R_r}
δ_{A_d}
δ_{E_d}
δ_{R_d}
δ_{R_d}
Σ
Mixer
$\boldsymbol{T}$

Figure 6.3: The PX4 control architecture adapted with disturbance observers which provide disturbance rejection.

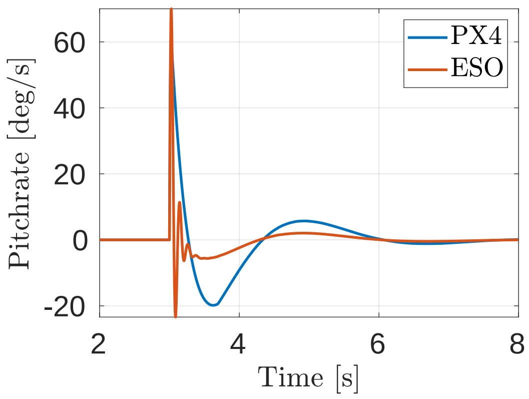


Figure 6.4: The pitch rate response of the multirotor being influenced by a step torque disturbance being rejected with either PX4 or ESO.

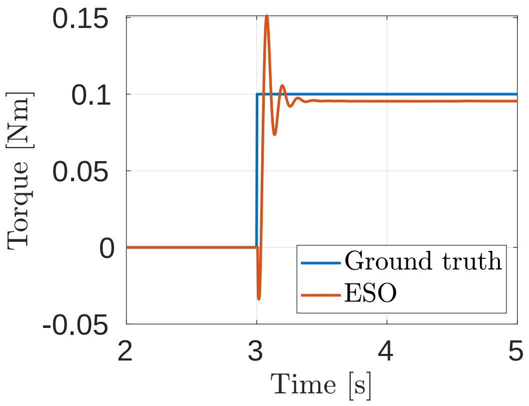


Figure 6.5: Estimation of a step torque disturbance in the $\bar{\boldsymbol{y}}_{\mathcal{B}}$ direction by a ESO during which it is being used in feedback.

6.2 Extended State Observer

In addition to normalising the estimated disturbance, it was required to reduce the value being subtracted to the pseudo control signal with a gain. If this was not done, the multirotor would become unstable. These gains are shown in Table 6.1 and were determined iteratively using step functions as the disturbance.

Fig. (6.4) shows the response of the multirotor being influenced by a torque disturbance simulated as a step function. As mentioned in Chapter 4, the high gains of the ESO and the low uncertainty in the linear model results in fast convergence to the estimated disturbance. Fig. (6.5) shows how the ESO estimates the disturbances. It is visible that the ESO overshoots the ground truth disturbance, and thus will introduce more disturbances. However, because of the fast dynamics of the angular rate controller, these overshoots are counteracted by the PX4 controller itself. Fig. (6.4) shows that the ESO provides significant improvements over the standard PX4 controllers, with some undesirably high frequency oscillation, before the ESO estimate reaches steady-state.

Fig. (6.6) shows the disturbance rejection of a step force disturbance influenc-

Control Loop	Gain
Angular Rate	0.4
Velocity	0.2

Table 6.1: Gains used for disturbance rejection in feedback loop when using ESO as estimator.

ing the multirotor. The slow converging speed to the ground truth disturbance and high overshoot of the estimated disturbance shown in Fig. (6.7) results in small improvements over the standard PX4 controllers. This poor performance is the result of the higher uncertainty in the linear velocity model of the multirotor. The high overshoot during estimation is not very prominent during feedback, because of the gain reducing its effect.

The poor performance of the ESO during the estimation of a step force disturbance provides little confidence for rejecting sinusoidal force disturbances. The estimation of sinusoidal force disturbances is shown in Fig. (6.9) with the response in Fig. (6.8). As expected, the ESO introduces more disturbances due to the poor estimation, but due to the gain reducing this effect, there is little to none improvement over the PX4 controller.

The ESO is capable of estimating torque sinusoidal disturbances, as shown in Fig. (6.11) with the response of the multirotor in Fig. (6.10). Even though the ESO estimates the sinusoidal disturbance accurately, it is not capable of rejecting it completely; however, it provides significant improvements over the PX4 controllers.

Since the ESO operates on SISO systems which lumps all the disturbances into one term, they cannot be used simultaneously for torque and force disturbances. Using them in both controller-loops will result in both ESO's estimating a disturbance, whether it is exclusively force or torque. This combined estimation results in the ESO's competing with each other and results in possible instability.

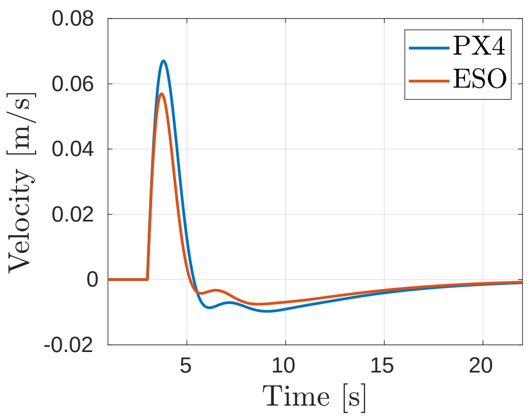


Figure 6.6: The response of the multirotor under the influence of a step force disturbance being rejected with PX4 or the ESO.

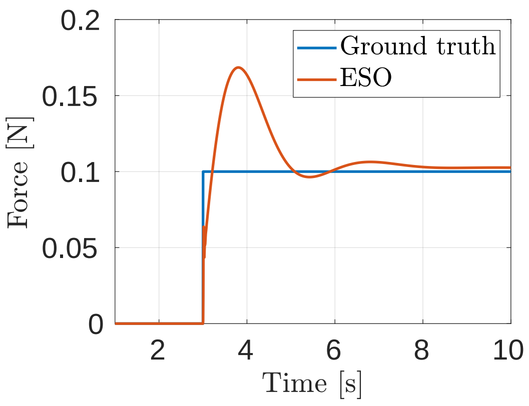


Figure 6.7: Estimation of a step force disturbance in the $\bar{\boldsymbol{x}}_{\mathcal{I}}$ direction by a ESO during which it is being used in feedback.

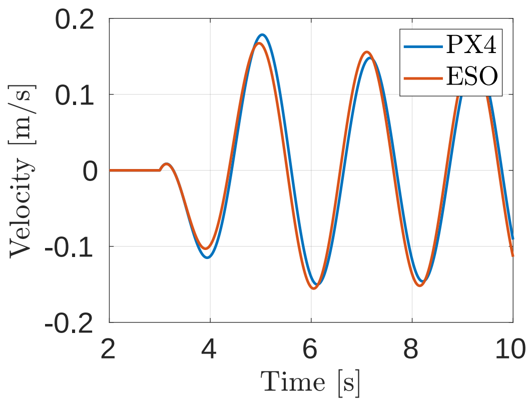


Figure 6.8: The response of the multirotor under the influence of a sinusoidal force disturbance being rejected with PX4 or the ESO.

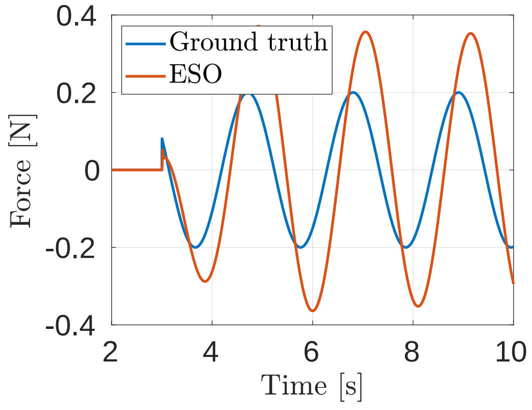


Figure 6.9: Estimation of a sinusoidal force disturbance in the $\bar{\boldsymbol{y}}_{\mathcal{B}}$ direction by the ESO during which it is being used in feedback.

6.3 Extended Kalman Filter

Fig. (6.13) displays the EKF estimating a force step disturbance and Fig. (6.12) the response of the multirotor. The EKF provides fast and little overshoot in its convergence towards the ground truth disturbance, and this translates to fast rejection of the disturbance with improvements over the standard PX4 controller. Fig. (6.16) shows that the use of the EKF provides small improvements when the multirotor is disturbed by a sinusoidal force disturbance, even though the estimation is closely matched to the ground truth disturbance as shown in Fig. (6.17).

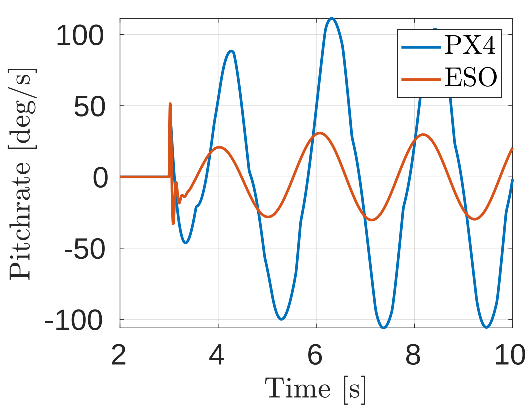


Figure 6.10: The response of the multirotor under the influence of a sinusoidal torque disturbance being rejected with PX4 or the ESO.

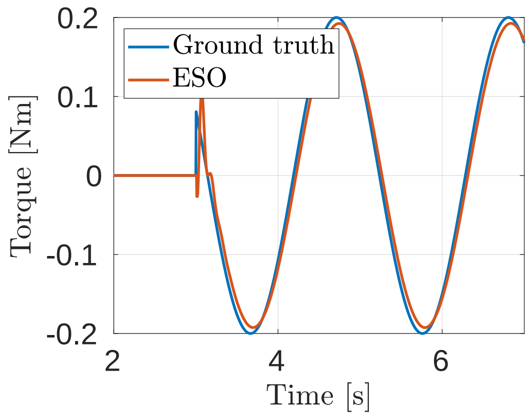


Figure 6.11: Estimation of a sinusoidal torque disturbance in the $\bar{\boldsymbol{y}}_{\mathcal{B}}$ direction by a ESO during which it is being used in feedback.

Fig. (6.14) shows the response of the multirotor experiencing a step torque disturbance and the estimation in Fig. (6.15). From the response, it is clear the EKF is capable of rejecting torque disturbances and the effect of the communication delay does not pose problems to the faster dynamics of the multirotor. The EKF proofs to show improvements over the PX4 controllers.

The estimation of a sinusoidal torque disturbance is shown in Fig. (6.18) with the response of the multirotor in Fig. (6.19). Though the EKF estimates the torque disturbance accurately, it does not reject the time-varying nature of the disturbance completely. It does again provide improvements over the PX4 controllers.

The EKF is capable of estimating all three force and torque disturbances in each respective direction and thus is more robust to disturbances in the various translation and attitude subsystems. However, as shown in Fig. (6.20), if the frequency of the sinusoidal disturbance passes some threshold, the EKF will start to perform worse than the PX4 controllers. This phenomenon is expected because the EKF can be viewed as an adaptive low pass filter which adapts based on the measurement and control updates as previously shown in Fig. (5.1) and explained in [43]. In Fig. (6.21) the estimation has a phase delay and a reduced magnitude from the ground truth, confirming this phenomenon. This threshold can be increased by increasing the variance of the disturbance model in Equation 5.1, but the effect of a noisier estimate might be underestimated.

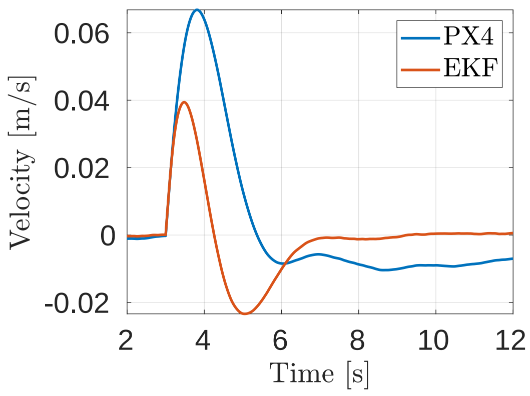


Figure 6.12: The response of the multirotor under the influence of a step force disturbance being rejected with PX4 or the EKF.

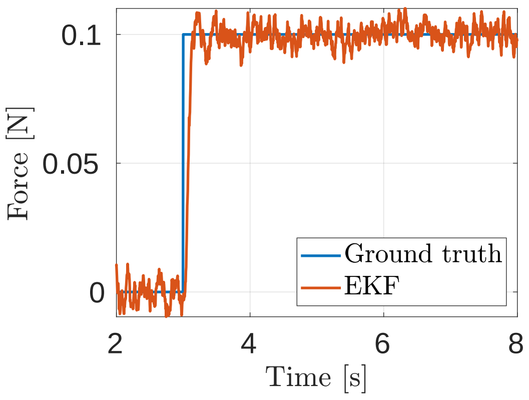


Figure 6.13: Estimation of a step force disturbance in the $\bar{\boldsymbol{y}}_{\mathcal{B}}$ direction by a EKF during which it is being used in feedback.

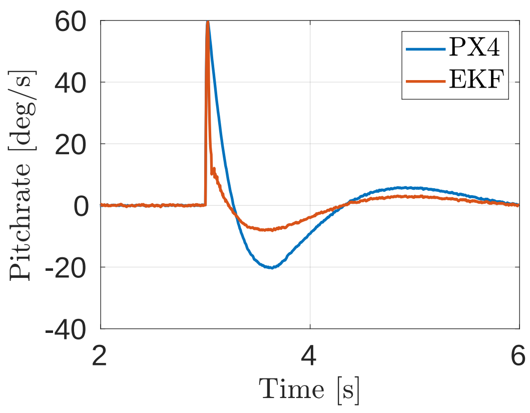


Figure 6.14: The response of the multirotor under the influence of a step torque disturbance being rejected with PX4 or the EKF.

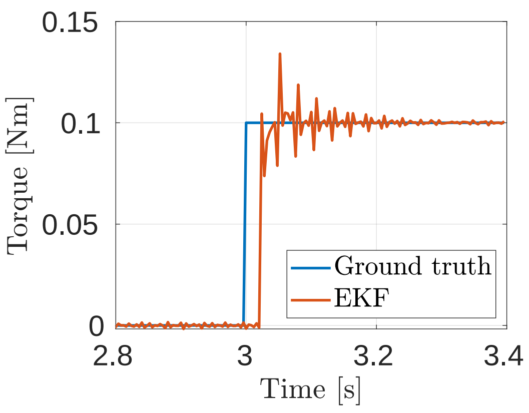


Figure 6.15: Estimation of a step torque disturbance in the $\bar{\boldsymbol{y}}_{\mathcal{B}}$ direction by a EKF during which it is being used in feedback.

6.4 Neural Network

Fig. (6.22) shows the response of the multirotor being influenced by a force disturbance. The use of the neural network shows improvements over the standard PX4 controller with the fast convergence speed and little to no overshoot for the estimation, as shown in Fig. (6.23).

The neural network provides improvements over the PX4 controller as shown in Fig. (6.24) with the estimation of the disturbance in Fig. (6.25) when rejecting sinusoidal force disturbances. Even though the neural network estimates the sinusoidal disturbance very closely with little offset, it does not reject the

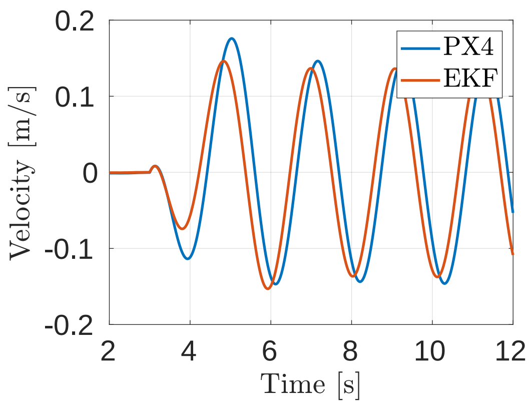


Figure 6.16: The response of the multirotor under the influence of a sinusoidal force disturbance being rejected with PX4 or the EKF.

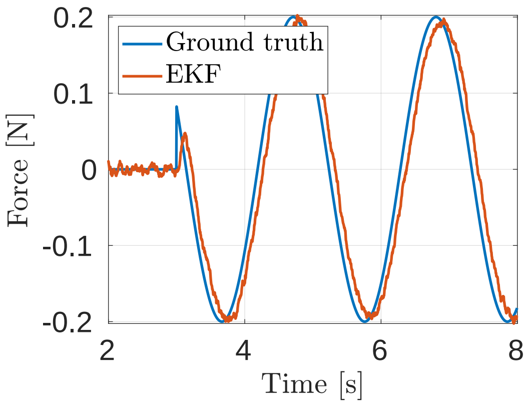


Figure 6.17: Estimation of a sinusoidal force disturbance in the $\bar{\boldsymbol{x}}_{\mathcal{I}}$ direction by a EKF during which it is being used in feedback.

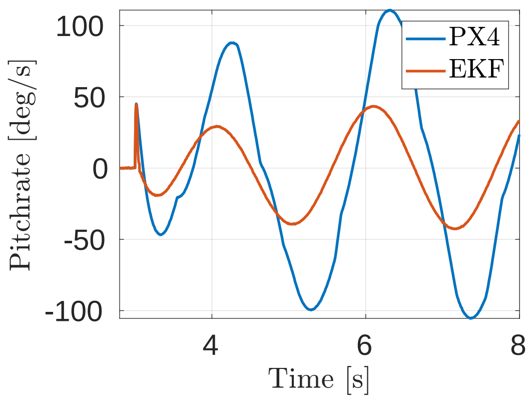


Figure 6.18: The response of the multirotor under the influence of a sinusoidal torque disturbance being rejected with PX4 or the EKF.

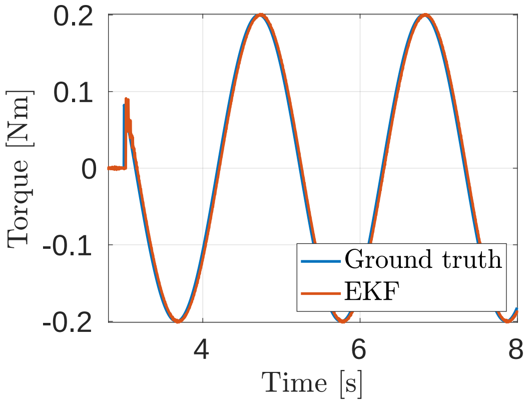


Figure 6.19: Estimation of a sinusoidal torque disturbance in the $\bar{\boldsymbol{y}}_{\mathcal{B}}$ direction by a EKF during which it is being used in feedback.

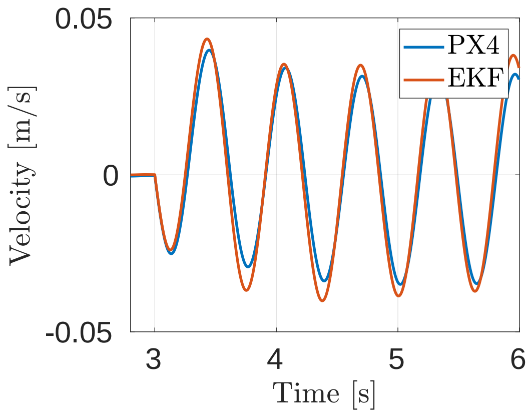


Figure 6.20: The response of the multirotor under the influence of a high frequency sinusoidal force disturbance being rejected with PX4 or the EKF.

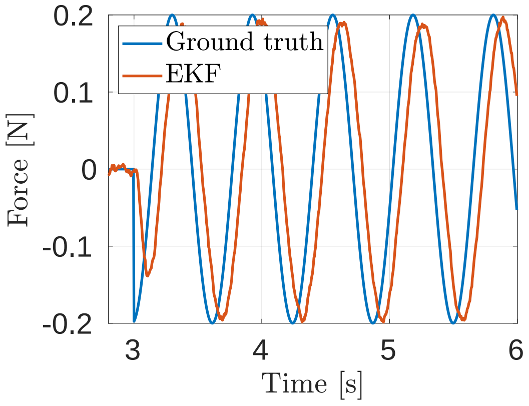


Figure 6.21: Estimation of a high frequency sinusoidal force disturbance in the $\bar{\boldsymbol{x}}_{\mathcal{I}}$ direction by a EKF during which it is being used in feedback.

disturbance entirely as would be expected, which was also seen with the EKF.

The dataset used to train the neural network only contained force disturbances and torque disturbances was not included. This was because of the required time taken to create such a dataset and the additional time required to train. The following results of torque disturbance rejection from a neural network is presented with the assumption that the neural network would perform to the same accuracy as that of force disturbances to represent some comparison to the other observers. Similar to the EKF, the neural network is capable of rejecting torque disturbance, as shown in Fig. (6.26) with the estimation in Fig. (6.27). The time delay does not cause any problems, and the use of the NN improves over the standard PX4 controllers. Again, similarly as the

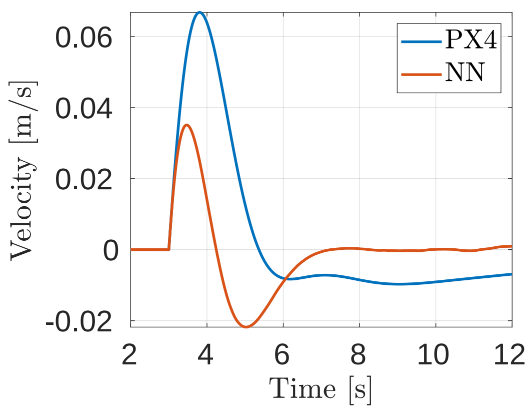


Figure 6.22: The response of the multirotor under the influence of a step force disturbance being rejected with PX4 or the NN.

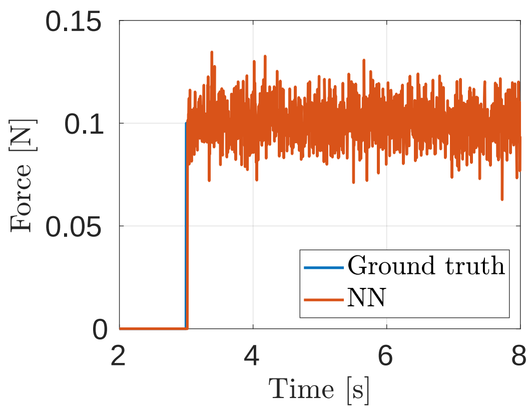


Figure 6.23: Estimation of a step force disturbance in the $\bar{\boldsymbol{x}}_{\mathcal{I}}$ direction by a NN during which it is being used in feedback.

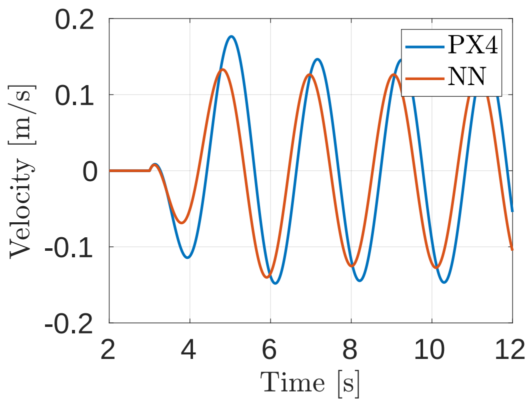


Figure 6.24: The response of the multirotor under the influence of a sinusoidal force disturbance being rejected with PX4 or the NN.

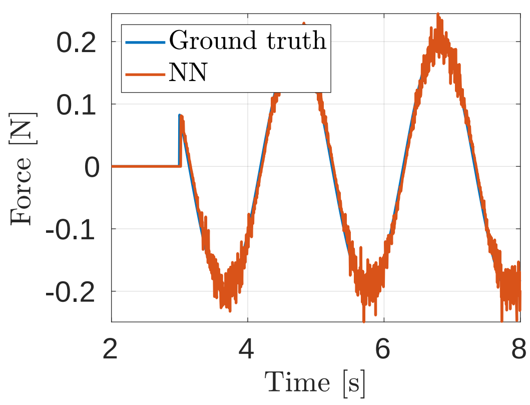


Figure 6.25: Estimation of a sinusoidal force disturbance in the $\bar{\boldsymbol{x}}_{\mathcal{I}}$ direction by a NN during which it is being used in feedback.

EKF, the NN is capable of estimating a torque sinusoidal disturbance shown in Fig. (6.29) with the response shown in Fig. (6.28).

6.5 Quantitative Comparisons Of Disturbance Rejection

The simulation environment in which the estimators will be compared is a multirotor at hover in steady-state. The multirotor must try to remain in this position during which force or torque disturbances described by step, ramp and sinusoidal signals is disturbing the multirotor. If the disturbance is a

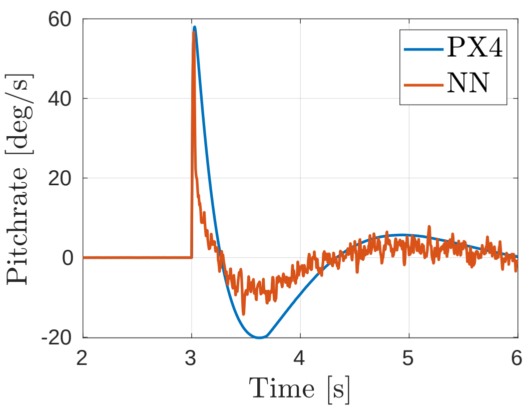


Figure 6.26: The response of the multirotor under the influence of a step torque disturbance being rejected with PX4 or the NN.

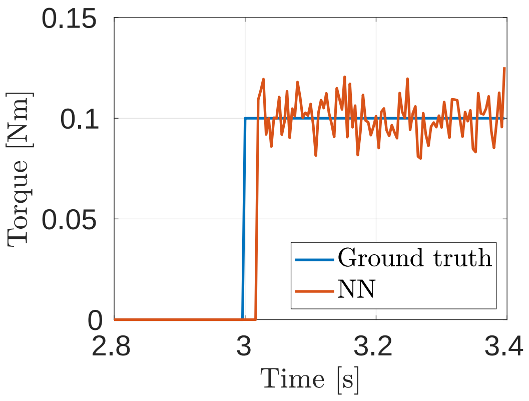


Figure 6.27: Estimation of a step torque disturbance in the $\bar{\boldsymbol{x}}_{\mathcal{I}}$ direction by a NN during which it is being used in feedback.

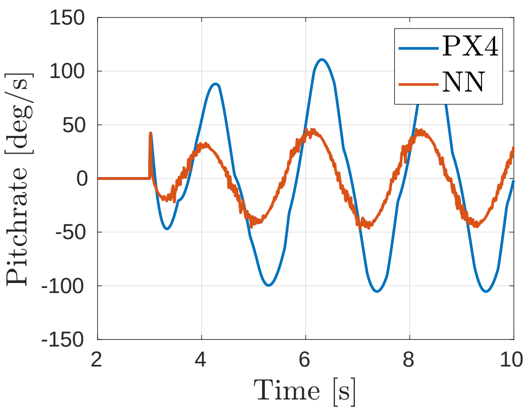


Figure 6.28: The response of the multirotor under the influence of a step sinusoidal disturbance being rejected with PX4 or the NN.

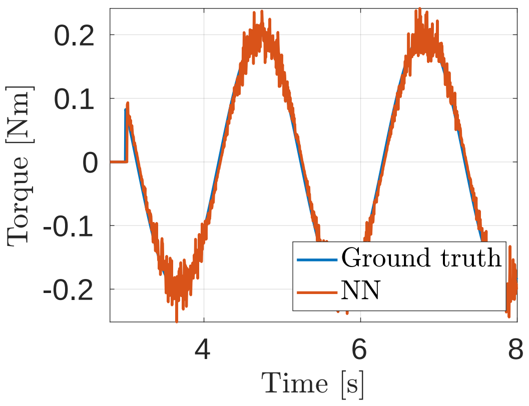


Figure 6.29: Estimation of a sinusoidal torque disturbance in the $\bar{\boldsymbol{x}}_{\mathcal{I}}$ direction by a NN during which it is being used in feedback.

force, it will influence the multirotor in the $\bar{\boldsymbol{x}}_{\mathcal{I}}$ direction and if it is a torque disturbance in the $\bar{\boldsymbol{y}}_{\mathcal{B}}$ direction. The decision to examine the behaviour of the multirotor from these two directions are based on the fact that the response from the $\bar{\boldsymbol{y}}_{\mathcal{I}}$ or $\bar{\boldsymbol{x}}_{\mathcal{B}}$ will yield the same results because of the symmetry of the multirotor. Furthermore, the disturbance is isolated to only one direction at a time and was chosen as such to provide certainty to the root causes of specific behaviour.

The performance of the estimators will be investigated using the following loss functions,

$$\text{IAE} = \int_{t_1}^{t_2} |e| dt \tag{6.9}$$

$$\text{ITAE} = \int_{t_1}^{t_2} t|e| dt \tag{6.10}$$

where e is the error signal and is defined as either the deviation from the steady-state $\bar{\boldsymbol{x}}_{\mathcal{I}}$ or $\bar{\boldsymbol{y}}_{\mathcal{B}}$ direction, depending on whether the disturbance is a force or torque. The two loss functions in Equations 6.9 and 6.10 penalises different characteristics of the response of the multirotor and is known as the Integrated Absolute Error (IAE) and Integrated Time Absolute Error (ITAE) respectively. IAE penalises both small and large errors equally, and if two scores are provided; the lower one is better. However, the IAE does not incorporate the time it takes to achieve steady-state, and ITAE implements this with the time variable, t in the integral. By using these two loss functions to compare different responses enable the designer to differentiate between estimators and to provide tangible commentary.

Tables 6.2 and 6.3 compares the various estimators and standard PX4 controllers using the IAE and ITAE loss functions during which disturbance of various forms are being rejected.

From Table 6.2, it is visible that the ESO performs the best to reject torque disturbances. This is mainly due to the reason of the ESO being designed as part of the controllers with no additional time-delay. However, the ESO performs the worst for rejecting force disturbance in comparison to the EKF and neural network for reasons previously discussed. The EKF and neural network performs the best for rejecting force disturbances with the neural network performing slightly better than the EKF. Since a communication delay is simulated, the EKF and neural network will perform better for estimating slower dynamics which manifest itself in the velocity plant. The EKF and neu-

Torque Disturbance	Step		Ramp		Sinusoidal	
	IAE	ITAE	IAE	ITAE	IAE	ITAE
PX4	0.466	1.881	0.392	2.510	23.38	333.2
ESO	**0.185**	**0.734**	**0.053**	**0.319**	**7.078**	**100.7**
EKF	0.333	1.807	0.142	1.245	13.91	198.9
NN	0.664	7.891	0.754	10.93	10.06	143.8

Table 6.2: Comparison of the various estimators and standard PX4 controllers being scored using the IAE and ITAE loss function for rejecting torque disturbances.

ral network performs better than the standard PX4 controllers rejecting torque disturbances and is perhaps a better overall implementation since the ESO can only be implemented in either velocity or angular rate control loops. Table 6.3 shows that the neural network performs worse than the PX4 controllers on step and ramp disturbance, but for sinusoidal disturbances, it performs the second best. The reason for performing worse on step and ramp disturbances is because during steady state, the neural network is still outputting noise around the ground truth value of the disturbance resulting in very small vibrations. These vibrations are penalised by the loss functions and is not a very good representation of its performance with respect to the other observers regarding torque disturbances.

As previously shown, the performance of the EKF reduces as the frequency of the disturbance increases. This should not be the case for the neural network if the dataset contained high frequency disturbances. The limitation would be the time delay due to the computation execution time of the neural network on the compute board.

From these results, it is evident that estimators being planned to execute on offboard devices should focus on rejecting disturbances which manifest in the slower dynamics of the system. In the multirotor case, this will be force disturbances. Rejecting torque disturbances in multirotors should occur on the flight control firmware running along the flight controller providing the high bandwidth required to compensate for the faster dynamics.

Force Disturbance	Step		Ramp		Sinusoidal	
	IAE	ITAE	IAE	ITAE	IAE	ITAE
PX4	0.177	1.371	0.246	2.288	2.018	28.63
ESO	0.133	1.043	0.191	1.780	2.052	29.30
EKF	0.073	0.396	0.066	0.485	1.928	27.64
NN	**0.065**	**0.368**	**0.059**	**0.471**	**1.742**	**25.00**

Table 6.3: Comparison of the various estimators and standard PX4 controllers being scored using the IAE and ITAE loss function for rejecting force disturbances.

6.6 Summary

The estimation of disturbances by the three observers was introduced into the disturbance rejection architecture to provide rejection of various types of disturbances. The disturbance rejection, which was provided by the ESO, EKF and neural network was presented and discussed. This was followed by comparing each of the three observers' disturbance rejection using a quantitative manner. From the results, it is evident that for torque disturbances acting on the multirotor, the ESO is much more suited to providing rejection. This is mainly due to the ESO executing at the same frequency as the inner controller providing the bandwidth required to reject the faster dynamics. The EKF and neural network are better suited to provide disturbance rejection to force disturbances acting on the velocity plant of the multirotor. The EKF and neural network is expected to be executing on a offboard device resulting in additional time delays which prefers the slower dynamics of the multirotor.

Chapter 7

Conclusion

This chapter will conclude the project by providing a summary of the project, which contains what has been achieved. This is followed by future work which originates from the results presented and finally the recommendation for work using this project as a foundation.

7.1 Summary of Project

Multirotors are being introduced in various industries, each with their own application-specific challenges to overcome for the multirotors to be of value in these sectors. These specific challenges can be described as disturbances influencing the multirotor due to the fact that these multirotors are required to operate outside of their original flight envelope.

The problem definition was formulated based on the following premise: if general disturbances can be rejected, it would provide a general solution towards all application-specific challenges. The proposed solution was formulated from the literature study on disturbance rejection for multirotors. It was selected as using neural networks to estimate these disturbances and reject them using a disturbance rejection architecture.

The system overview was presented, which explained the workflow of the project at a high level. The system overview contained the simulation environment namely Gazebo, the flight controllers used which is PX4, ROS which is used for the communication between system components and then software such as MATLAB and Tensorflow used for testing and implementation.

Sim2Real techniques were implemented in the simulation environment to improve the neural network's ability to perform well on real data and not only the simulated data on which it was trained. This technique influenced the following: simulation environment, the neural network architecture, the flight path which the multirotor attempts to follow, the construction of the dataset and how the disturbances occur in simulation.

Two other classical methods used for disturbance estimation were presented and their estimation of various disturbances discussed. These two classical estimation techniques are the Extended Kalman Filter and the Extended State Observer. The results showed that the EKF and ESO are both capable of estimating torque and force disturbances which are of a time-varying nature and not.

The estimation of the trained neural network was presented on the validation dataset and then was followed by the estimation of practical flight tests. The estimation of disturbances of practical flight tests showed that the neural network had transferred acceptably to reality. This is showed by the neural network estimating little to no disturbance during a practical flight which the multirotor was carrying no payload. In the other practical flight tests presented during which the multirotor carried a payload of 0.14kg, the neural network predicted a payload of 0.18kg. These results verify that the neural network successfully transferred from simulation to reality.

The project was concluded by comparing the rejection of disturbances the three observers provide in a MATLAB environment. The three observers were scored based on the IAE and ITAE loss functions. It showed the Extended State Observer performs the best for torque disturbances whereas the Extended Kalman Filter and the neural network perform better on slower dynamics such as force disturbances. Specifically, the use of a neural network as an observer in a disturbance rejection architecture shows compelling evidence as oppose to the classical observers as the method for rejecting unknown disturbances influencing a multirotor.

7.2 Future Work

The loss function during training of the neural network showed that the neural network has underfitted the training dataset. Further training is thus required and would improve the estimation of the neural network. The results of the validation dataset showed some large errors made by the neural network and using the MSE loss function to punish larger errors might improve its results.

The disturbance rejection of torque disturbances from a neural network was presented although the training of the neural network only contained force disturbances. This was done to provide comparison to the other observers with the assumption that the neural network would achieve similar accuracy.

Thus, to verify the results shown torque disturbance should be contained in the dataset.

The neural network showed promising results when the estimated disturbances was used in feedback, but this was tested in a MATLAB environment without the possible side-effects that the neural network portrayed. To verify the disturbance rejection results, the neural network must be tested practically. This will entail that the neural network must be executing on a companion board alongside the flight controllers during which controlled disturbances must be induce onto the multirotor to compare the estimation and the effects of it being used in feedback.

7.3 Recommendation

For practical testing of using a trained neural network alongside the PX4 flight controllers for providing disturbance rejection the ROS2 communication solution should be used. ROS2 uses a Real Time Publish Subscribe (RTPS) communication protocol which provides timing guarantees for the critical control loops.

Tensorflow is one of many software packages used to train and validate neural networks, however during the project another machine learning software namely, PyTorch, has increased in popularity. There were challenges to using Tensorflow which caused delays during training, thus to investigate what machine learning software provides the best support is advised.

Bibliography

[1] Veličković, P.: Complete collection of my PGF/TikZ figures. 2016.
Available at: `https://github.com/PetarV-/TikZ`

[2] Gazebo simulation. `https://dev.px4.io/v1.9.0/en/simulation/gazebo.html`, 2020. Accessed: 2020-09-30.

[3] Aerobotics | Press. 2020.
Available at: `https://www.aerobotics.com/press/aerobotics-introduces-latest-yield-estimation-technology-for-growers`

[4] Oil and Gas | Airborne Drones. .
Available at: `https://www.airbornedrones.co/oil-and-gas/`

[5] Benefits of using drones in disasters | Airborne Drones. .
Available at: `https://www.airbornedrones.co/airborne-drones-reports-on-in-depth-study-reveals-how-drones-can-help-in-all-phases-of-a-disaster/`

[6] First U.S. FAA-approved 'beyond-line-of-sight' drone flight completed | Reuters. 2020.
Available at: `https://www.reuters.com/article/us-usa-faa-drones-idUSKCN1US2LR`

[7] Sharma, P. and Singh, A.: Era of deep neural networks: A review. In: *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pp. 1–5. 2017.

[8] Kotzé, H., Jordaan, W. and Kamper, H.: Training neural networks for control, estimation and disturbance rejection. *Elsevier, IFAC Papers Online*, 2020.

[9] Smeur, E.J.J., de Croon, G.C.H.E. and Chu, Q.: Cascaded Incremental Nonlinear Dynamic Inversion Control for MAV Disturbance Rejection. jan 2017. 1701.07254.
Available at: `http://arxiv.org/abs/1701.07254`

[10] Fernandez, R.A.S., Dominguez, S. and Campoy, P.: L1 adaptive control for Wind gust rejection in quad-rotor UAV wind turbine inspection. In: *2017*

International Conference on Unmanned Aircraft Systems (ICUAS), pp. 1840–1849. IEEE, jun 2017. ISBN 978-1-5090-4495-5.
Available at: `http://ieeexplore.ieee.org/document/7991485/`

[11] Celen, B. and Oniz, Y.: Trajectory Tracking of a Quadcopter Using Fuzzy Logic and Neural Network Controllers. In: *2018 6th International Conference on Control Engineering & Information Technology (CEIT)*, pp. 1–6. IEEE, oct 2018. ISBN 978-1-5386-7641-7.
Available at: `https://ieeexplore.ieee.org/document/8751810/`

[12] Al-Mahasneh, A.J., Anavatti, S.G., Ferdaus, M. and Garratt, M.A.: Adaptive Neural Altitude Control and Attitude Stabilization of a Hexacopter with Uncertain Dynamics. In: *2019 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*, pp. 44–49. IEEE, jul 2019. ISBN 978-1-7281-3745-2.
Available at: `https://ieeexplore.ieee.org/document/8784844/`

[13] Koch, W., Mancuso, R., West, R. and Bestavros, A.: Reinforcement Learning for UAV Attitude Control. *ACM Transactions on Cyber-Physical Systems*, vol. 3, no. 2, pp. 1–21, feb 2019. ISSN 2378962X.
Available at: `http://dl.acm.org/citation.cfm?doid=3284746.3301273`

[14] Vankadari, M.B., Das, K., Shinde, C. and Kumar, S.: A Reinforcement Learning Approach for Autonomous Control and Landing of a Quadrotor. In: *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 676–683. IEEE, jun 2018. ISBN 978-1-5386-1354-2.
Available at: `https://ieeexplore.ieee.org/document/8453468/`

[15] Hwangbo, J., Sa, I., Siegwart, R. and Hutter, M.: Control of a Quadrotor With Reinforcement Learning. *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, oct 2017. ISSN 2377-3766.
Available at: `http://ieeexplore.ieee.org/document/7961277/`

[16] Dai, B., He, Y., Zhang, G., Gu, F., Yang, L. and Xu, W.: Wind Disturbance Rejection for Unmanned Aerial Vehicle Based on Acceleration Feedback Method. In: *2018 IEEE Conference on Decision and Control (CDC)*, pp. 4680–4686. IEEE, dec 2018. ISBN 978-1-5386-1395-5.
Available at: `https://ieeexplore.ieee.org/document/8619798/`

[17] Bin Qiu, Hejin Xiong and Jian Fu: The position control of micro quad-rotor UAV based on ADRC. In: *2015 Chinese Automation Congress (CAC)*, pp. 422–426. IEEE, nov 2015. ISBN 978-1-4673-7189-6.
Available at: `http://ieeexplore.ieee.org/document/7382537/`

[18] Zhang, R., Quan, Q. and Cai, K.Y.: Attitude control of a quadrotor aircraft subject to a class of time-varying disturbances. *IET Control Theory and Applications*, 2011. ISSN 17518644.

[19] Suhail, S.A., Bazaz, M.A. and Hussain, S.: Altitude and attitude control of a quadcopter using linear active disturbance rejection control. In: *2019 International Conference on Computing, Power and Communication Technologies (GUCON)*, pp. 281–286. Sep 2019. ISSN null.

[20] Zhao, Y., Cao, Y. and Fan, Y.: Disturbance observer-based attitude control for a quadrotor. In: *2017 4th International Conference on Information, Cybernetics and Computational Social Systems (ICCSS)*, pp. 355–360. July 2017. ISSN null.

[21] Hentzen, D., Stastny, T., Siegwart, R. and Brockers, R.: Disturbance estimation and rejection for high-precision multirotor position control. 2019. `1908.03166`.

[22] Verberne, J. and Moncayo, H.: Robust Control Architecture for Wind Rejection in Quadrotors. In: *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 152–161. IEEE, jun 2019. ISBN 978-1-7281-0333-4.
Available at: `https://ieeexplore.ieee.org/document/8798039/`

[23] Xiang, T., Jiang, F., Hao, Q. and Cong, W.: Adaptive flight control for quadrotor UAVs with dynamic inversion and neural networks. In: *2016 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pp. 174–179. IEEE, sep 2016. ISBN 978-1-4673-9708-7.
Available at: `http://ieeexplore.ieee.org/document/7849485/`

[24] Bisheban, M. and Lee, T.: Geometric Adaptive Control for a Quadrotor UAV with Wind Disturbance Rejection. In: *2018 IEEE Conference on Decision and Control (CDC)*, pp. 2816–2821. IEEE, dec 2018. ISBN 978-1-5386-1395-5.
Available at: `https://ieeexplore.ieee.org/document/8619390/`

[25] Bari, S., Zehra Hamdani, S.S., Khan, H.U., ur Rehman, M. and Khan, H.: Artificial Neural Network Based Self-Tuned PID Controller for Flight Control of Quadcopter. In: *2019 International Conference on Engineering and Emerging Technologies (ICEET)*, pp. 1–5. IEEE, feb 2019. ISBN 978-1-7281-0278-8.
Available at: `https://ieeexplore.ieee.org/document/8711864/`

[26] Shi, G., Shi, X., O'Connell, M., Yu, R., Azizzadenesheli, K., Anandkumar, A., Yue, Y. and Chung, S.-J.: Neural Lander: Stable Drone Landing Control Using Learned Dynamics. In: *2019 International Conference on Robotics and Automation (ICRA)*, pp. 9784–9790. IEEE, may 2019. ISBN 978-1-5386-6027-0.
Available at: `https://ieeexplore.ieee.org/document/8794351/`

[27] Allison, S., Bai, H. and Jayaraman, B.: Wind Estimation Using Quadcopter Motion: A Machine Learning Approach. jul 2019. `1907.05720`.
Available at: `http://arxiv.org/abs/1907.05720`

[28] Bannwarth, J.X.J., Chen, Z.J., Stol, K.A. and MacDonald, B.A.: Disturbance accomodation control for wind rejection of a quadcopter. In: *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 695–701. IEEE, jun 2016. ISBN 978-1-4673-9334-8.
Available at: `http://ieeexplore.ieee.org/document/7502632/`

[29] Matus-Vargas, A., Rodriguez-Gomez, G. and Martinez-Carranza, J.: Aerodynamic disturbance rejection acting on a quadcopter near ground. In: *2019 6th International Conference on Control, Decision and Information Technologies, CoDIT 2019*. 2019. ISBN 9781728105215.

[30] Jiang, F., Pourpanah, F. and Hao, Q.: Design, Implementation and Evaluation of a Neural Network Based Quadcopter UAV System. *IEEE Transactions on Industrial Electronics*, pp. 1–1, 2019. ISSN 0278-0046.
Available at: `https://ieeexplore.ieee.org/document/8676108/`

[31] OpenAI, Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L. and Zaremba, W.: Learning dexterous in-hand manipulation. 2018. `1808.00177`.

[32] Erasmus, A.: *Stabilization of a rotary wing unmanned aerial vehicle with an unknown suspended payload*. Master's thesis, Stellenbosch University, 2020.

[33] Goodfellow, I., Bengio, Y. and Courville, A.: *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[34] Bengio, Y., Simard, P. and Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

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

[36] Krogh, A. and Hertz, J.A.: A Simple Weight Decay Can Improve Generalization. 1992.
Available at: `https://papers.nips.cc/paper/563-a-simple-weight-decay-can-improve-generalization`

[37] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R.: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.
Available at: `http://jmlr.org/papers/v15/srivastava14a.html`

[38] Brescianini, D., Hehn, M. and D'Andrea, R.: Nonlinear quadrocopter attitude control. technical report. 2013.

[39] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems. 2015. Software available from tensorflow.org.
Available at: `https://www.tensorflow.org/`

[40] Kaspar, M., Osorio, J.D.M. and Bock, J.: Sim2real transfer for reinforcement learning without dynamics randomization. 2020. 2002.11635.

[41] Borrego, J., Figueiredo, R., Dehban, A., Moreno, P., Bernardino, A. and Santos-Victor, J.: A generic visual perception domain randomisation framework for gazebo. In: *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pp. 237–242. April 2018.

[42] Kingma, D.P. and Ba, J.: Adam: A method for stochastic optimization. 2014. 1412.6980.

[43] van Daalen, C. and Jones, T.: Advance estimation 813: Course notes v(0.4). February 2019.

[44] Offboard px4 v1.9.0 user guide. 2020.
Available at: https://docs.px4.io/v1.9.0/en/flight_modes/offboard.html

[45] Yang, M., Lang, X., Long, J. and Xu, D.: Flux immunity robust predictive current control with incremental model and extended state observer for pmsm drive. *IEEE Transactions on Power Electronics*, vol. 32, no. 12, pp. 9267–9279, 2017.

[46] Yao, J. and Deng, W.: Active disturbance rejection adaptive control of hydraulic servo systems. *IEEE Transactions on Industrial Electronics*, vol. 64, no. 10, pp. 8023–8032, 2017.

[47] Yang, L., Zhang, M., Tian, L. and Wang, P.: Improved phase plane attitude control of space vehicles with extended state observer considering disturbance and thruster dynamics. In: *2020 Chinese Control And Decision Conference (CCDC)*, pp. 4417–4422. 2020.

[48] Zheng, S.: Topic 14: Unbiased Estimation *. Tech. Rep., Missouri State University, 2011.
Available at: http://people.missouristate.edu/songfengzheng/Teaching/MTH541/Lecturenotes/evaluation.pdf