# Estimating Unknown Dynamics of a Quadrotor Aerial Vehicle Using Artificial Neural Networks

by

©Sachithra H. Atapattu

A Thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements for the degree of

**Master of Engineering**

Memorial University of Newfoundland

**October 2020**

St. John's Newfoundland

# Abstract

This thesis presents the use of artificial neural networks (ANN) to accurately estimate the unmodelled dynamics of a quadrotor aerial vehicle. Due to the complexity of aerodynamic models, ground effects, actuator non-linearities, and disturbances that occur during operation, usually a complete dynamic modelling of a quadcopter is difficult to achieve. A quadcopter's flight stability mainly depends on its control system and its navigation system. Therefore, the knowledge of the platform's unknown dynamic model will allow enhanced design of the autonomous system. This thesis focuses on estimating the unmodelled dynamics using machine learning (ML) techniques to achieve better design of control and navigation systems of quadcopters.

In this work, the dynamic behaviour of a quadcopter is expressed as a combination of a nominal model and an unknown model. The nominal model is derived from classical rigid-body dynamic equations, while ML approaches are used to estimate the unknown model. Among the available ML techniques, two commonly used approaches; Gaussian process regression (GPR) and ANN are evaluated in this thesis.

Training and testing of the methods were done offline, using a dataset gathered by manually flying an AscTec Hummingbird quadcopter in an OptiTrack motion capture environment. For the comparison of the selected ML techniques, the disturbance force was learnt initially. Since ANN showed better results than GPR in initial comparison, ANN was chosen and further evaluated with different architectures. The disturbances in force and torque were learnt separately using two independent ANNs. These estimated residual dynamics were added to the nominal dynamic model to obtain the compound dynamics. The results demonstrate reduction in model error in comparison to the nominal model.

To demonstrate the use of learnt residual dynamics in improved controller design, a trajectory tracking controller was implemented in a simulated environment. This simulation was designed to track a desired trajectory in the presence of added disturbances estimated using the ANN model. The results are compared with the trajectory tracking results obtained with only the nominal model. As expected, the controller with disturbance correction using ANN model showed improved performance.

**keywords:** Artificial neural networks, Gaussian process regression, Quadcopter dynamics learning

# Acknowledgements

I would like to express my sincere gratitude to my thesis supervisors, Drs. Oscar De Silva, George K. I. Mann, and Raymond G. Gosine for the guidance for completion of my Master of Engineering thesis. Their invaluable lessons and motivation throughout the research helped me to achieve success in my research.
I would like to acknowledge the funding agencies, NSERC Discovery grants, Dr. Gosine VP academic grant and Research Chair grant for providing the financial support.
I am privileged to be a member in the Intelligent Systems Laboratory (IsLab) with the assistance of Dr. Thumeera R. Wanasinghe, Mr. Mihiran Galagedarage Don, Mr. Eranga Fernando, Mr. Mahmoud Abd El Hakim, Mr. Kusal Tennakoon and Mr. Didula Dissanayake. I am specially grateful to Mr. Eranga Fernando for his help when capturing the data set using the quadcopter.
My heartfelt thanks go to my husband, Mr. Nushen Senevirathna (who is also a member of the IsLab) for always being by my side to support my work.

# Table of Contents

# List of Tables

# List of Figures

# Abbreviations

**ANN** Artificial Neural Networks

**API** Application programming interface

**BLR** Bayesian linear regression

**CG** Center of Gravity

**DOF** Degrees of Freedom

**ESKF** Error-state Kalman filter

**GP** Gaussian Process

**GPR** Gaussian Process Regression

**IMU** Inertial measurement unit

**LM** Levenberg-Marquardt

**LQR** Linear Quadratic Regulator

**MAV** Multi-rotor Aerial Vehicle

**ML** Machine Learning

**MODERNN** MOdular DEep Recurrent Neural Network

**MPC** Model Predictive Control

**NN** Neural Networks

**PD** Proportional-Derivative

**PID** Proportional-Integral-Derivative

**ReLU** Rectified Linear Units

**RMSE** Root-mean-square error

**SSE** Sum of squared errors

**UAV** Unmanned Aerial Vehicle

**wBLR** weighted Bayesian linear regression

# Chapter 1

# Introduction

This chapter includes the motivation of this research followed by an introduction to quadcopter dynamic models and disturbance models. Then an overview of the available methods to estimate unknown dynamics is presented. In the final sections of this chapter, the problem statement is formulated and the objectives of this research are highlighted.

## 1.1 Motivation

Quadcopter is a type of unmanned aerial vehicle (UAV) which has become popular in robotics during past decades due to its hovering capability, vertical takeoff and landing ability, better maneuverability, flexible design, easy construction, and low cost [2, 3]. Quadcopters have been used in a wide range of indoor and outdoor applications such as surveillance, aerial photography, transportation, and inspection tasks [2, 4]. In addition, the applications of quadcopters are further growing into areas like military operations and search and rescue missions in hazardous environments for humans [3].

A quadcopter has only four rotors as actuators which can be used to control thrust, roll, pitch and yaw independently. However, the control demand of the 6 degrees of freedom (DOF) as a rigid-body in a Euclidean space makes the system underactuated [5]. It is worth mentioning here that although there are multi-rotor aerial vehicles (MAVs) with six or more rotors, due to the planar configuration of rotors, they are also considered underactuated. Hence, the stable operation of a quadrotor aerial vehicle highly relies on its control system [4].

A controller generally attempts to drive the system to a desired state by applying control inputs, with the objective of minimizing the error between desired and actual states. Although simple feedback controllers such as Proportional-Integral-Derivative (PID) controllers are capable of minimizing the state error, to some degree, model-based controllers outperform in complex non-linear systems. In model-based controllers, majority of the error components are captured by the model. Therefore, the state error is reduced, allowing the controller to converge faster and more accurately.

The performance of model-based controllers such as Linear Quadratic Regulators (LQR) and Model Predictive Control (MPC) directly depend on the accuracy of the model [6], requiring an accurate dynamic model to achieve improved controller performance. However, most of the time, the complex dynamic interactions are usually neglected, in order to reduce analytical and computational complexity. Furthermore, the system imperfections and unmodelled biases present in practical systems add-up to this discrepancy [6]. Hence, the modelled dynamics of a physical system is a simplified version of the underlying dynamics and under-represents the real behaviour of the system.

In addition, the system dynamics are changed considerably when attaching a payload, which is essential in some of the applications such as package delivery, surveillance

and inspection tasks [2]. When using the same control parameters throughout these changing dynamics, it can cause system instabilities or reduced performance.

Generally, these system imperfections are handled by incorporating robust controllers, rather than estimating the unknown dynamics [3, 7]. But, tuning the controller to safely handle a wide range of possible scenarios can be difficult and may not utilize full system capabilities. Therefore, estimating the unknown dynamics will be beneficial to improve the quadcopter's flight performance.

Estimating the unknown dynamics using traditional methods is challenging since some of these dynamics can be caused by unknown sources. Typically, machine learning (ML) techniques only consider the observed co-relation between the input and the output. Therefore, incorporating ML techniques to estimate these unknown dynamics can be considered as a better approach.

Recently, ML approaches have become popular in estimating the unknown dynamics of quadcopters [6]. Few of the ML techniques that are commonly used in literature are neural networks [4, 8], Gaussian process regression [6], and Bayesian linear regression [9]. Work in [4, 6, 8, 9] have demonstrated that the performance of the controller is improved when the learnt dynamics are included in the controller model.

This thesis proposes the use of ML methods to achieve enhanced control of quadcopters by capturing the unmodelled components in force and torque models of the vehicle.

## 1.2 Dynamics of a Quadcopter

Modelling the dynamics of a quadcopter is challenging because of the complex non-linear aerodynamics forces, gyroscopic moments, coupling between translational and rotational dynamics leading to complex non-linear dynamic models [3]. However, its basic dynamics can be derived from first principles using Newton-Euler laws, as shown in (1.1) and (1.2) [8, 10–12].

Let the superscripts and subscripts $w$ and $b$ denote the world reference frame and the quadcopter body reference frame, respectively. Then, ${}^{w}\boldsymbol{R}_{b}$ is the attitude rotation matrix from body frame to world frame.

$$m\boldsymbol{a} = m\boldsymbol{g} + {}^{w}\boldsymbol{R}_{b}\boldsymbol{T} \tag{1.1}$$

$$\boldsymbol{J}\boldsymbol{\alpha} = (\boldsymbol{J}\boldsymbol{\omega}) \times \boldsymbol{\omega} + \boldsymbol{\tau} \tag{1.2}$$

Here, $m$ and $\boldsymbol{J}$ are the mass and inertia tensor of the quadcopter, $\boldsymbol{g} = [0, 0, -9.81]^{T}$ is the gravity vector expressed in world frame, $\boldsymbol{a}$ is the acceleration of body frame relative to world frame, expressed in world frame, $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$ are the angular acceleration and angular velocity of body frame relative to world frame, expressed in body frame, and finally, $\boldsymbol{T}$ and $\boldsymbol{\tau}$ are the total body thrust and torques with respect to world frame, expressed in body frame. The operator $\times$ indicates the cross product between $\boldsymbol{J}\boldsymbol{\omega}$ and $\boldsymbol{\omega}$. These equations will be further discussed in chapter 4.1.

## 1.3 Unmodelled Dynamics and Imperfections

The above equations capture the basic dynamics of a quadcopter assuming ideal conditions. But in practical scenarios, there are several disturbance sources that can affect the dynamics of a quadcopter. One such disturbance is the aerodynamic drag force $\boldsymbol{F}_d$, which can be approximated as proportional to the body-frame velocity $\dot{\boldsymbol{p}}_b$ as shown in (1.3), where $\boldsymbol{p}_b$ is the position of the quadcopter relative to the world frame [13] and $\boldsymbol{K}_d$ is the drag coefficient matrix.

$$\boldsymbol{F}_d = -\boldsymbol{K}_d \dot{\boldsymbol{p}}_b, \tag{1.3}$$

This drag force can be modelled if the drag coefficients are estimated.

Another effect usually ignored yet significant is the physical imbalances of the quadcopter, i.e., estimated center of gravity (CG) can be different from the nominal point, which is usually considered to be located at the geometric center. This can also affect the rotational inertia values, which are usually expressed at the geometric center. Such an imbalance of the quadcopter can happen due to several reasons, e.g. when payloads are attached to the quadcopter, modifications are made to the quadcopter, or due to degradation of the quadcopter during its operation lifespan. Additionally, there can be on-board sensor transformation errors and bias components of the sensors, which might be neglected without capturing in the nominal dynamic equations. In practical applications, even though the same motors and propellers are used, there can be manufacturing imperfections and differences in motors and propellers. In a basic dynamic model, all motors and propellers are assumed to be having the same characteristics, but they may not. Therefore, it can also result in significant differences in the dynamics than the actual model.

Although a nominal model neglects the said complex interactions, several research work have been carried out to robustly control the quadrotor in the presence of unmodelled disturbances [2, 7, 14]. These robust control algorithms handle the unmodelled dynamics of quadcopters. Quadcopter has an on-board attitude controller or an angular rate controller (most of the time a PID controller) which can be tuned to achieve the desired control [2]. In addition to this low-level roll-pitch-thrust controller, quadcopters generally have a high-level controller which controls the position, velocity and yaw states.

In order to compensate for the unmodelled dynamics, adaptive control algorithms are regarded as state-of-the-art solutions since they can track a desired output when there are parametric uncertainties [2, 7, 14]. Adaptive control has been incorporated in [2, 7, 14] to improve the performance of the quadcopter attitude controllers when manipulating different payloads. They have presented simulated and experimental validation of the consistent controller performance, and enhanced system robustness when incorporating adaptive control in the quadcopter controllers. Even though adaptive control algorithms have proven results in safe control in the presence of disturbances, their response is slow and has delayed feedback [8]. Therefore, to achieve faster response and improve the flight performance, estimation of unmodelled dynamics can be included in the dynamic model of the controller [8]. Then, the complete dynamics of a quadcopter is written as a combination of known model and an unknown model as shown in (1.4) [6]. The goal of detailed modelling is to estimate this unknown model $g$ as a function of states $\boldsymbol{x}$ and control inputs $\boldsymbol{u}$. Here, $k$ denotes the $k^{th}$ time step.

$$\boldsymbol{x}_{k+1} = \underbrace{f(\boldsymbol{x}_k, \boldsymbol{u}_k)}_{\text{Known model}} + \underbrace{g(\boldsymbol{x}_k, \boldsymbol{u}_k)}_{\text{Unknown model}} \tag{1.4}$$

Traditional parametric regression methods can be considered as the basic approach to model these unknown dynamics. Linear regression is the simplest classical approach, but due to the dimensionality and non-linearities of these unknown interactions, linear regression may not capture the errors accurately, resulting in poor system performance [3]. Non-linear parametric regression techniques can be better than linear regression, since the operating range of the model is fitted to a non-linear function in these methods. But, to identify a suitable model or a function, the nature of the unmodelled dynamics should be known beforehand. Furthermore, the model may not follow the same function throughout the operating region, resulting in over-fitting or under-fitting in some sections. In order to overcome this issue, more advanced methods are introduced in the literature. Recently, sophisticated ML techniques have become a common strategy for model regression in the area of learning dynamics [9]. Literature shows that ML techniques have been successfully used to learn unmodelled dynamics as well as learn controller parameters [6, 8, 9].

## 1.4 Estimation of Unknown Dynamics and Imperfections

As mentioned in section 1.3, there are ML techniques that can be used to estimate the unknown dynamics and disturbances that are not captured in the quadcopter's nominal dynamic model. Among these methods, Bayesian linear regression (BLR), Gaussian process regression (GPR) and different versions of artificial neural network (ANN) techniques are the most commonly used approaches [15]. A theoretical comparison of these approaches was carried out in this section to determine the most suitable approaches to adapt in this work.

Neural networks (NN) is a classical approach for model regression in machine learning [16]. The accuracy of an ANN model can be increased by training for a large dataset [17]. In addition to that, NN approach is capable of learning models that are highly non-linear, due to its flexible structure. The complexity of the system can be achieved by increasing the number of layers and neurons in the NN architecture [18]. Most importantly, compared to GPR, NN requires a lesser computation time and memory for larger training sets [15]. Considering the purpose of this work, i.e., learning unknown quadcopter dynamics, most of these properties will be beneficial. To obtain a highly accurate model and to capture higher order non-linearities, a large number of neurons should be used in the NN. A larger training set is needed to serve this purpose. If the training set is not sufficient for the network dimensionality, NNs can result in ill-trained models causing unpredictable outputs and instabilities [8]. In our case, this can be easily overcome by collecting a sufficiently large training dataset.

This thesis also considers GPR as a suitable machine learning candidate for quadcopter unknown model identification. It is a probabilistic non-parametric regression approach, which is shown to outperform NN in some circumstances [6]. An advantage in these probabilistic models is that they provide the model uncertainty which can be used to calculate the upper and lower confidence bounds of the prediction function [9]. In GPR, a normally distributed random variable is assigned to every point in the state-space. Hence, the model can be described in a probabilistic framework [19]. Similar to ANN, the accuracy of the GPR depends on the density of the dataset used for training. However, GPR will require higher computational power and time compared to ANN when a larger dataset is used [9]. Despite these drawbacks, GPR is also proven as a successful method in learning unknown quadcopter dynamics models and controllers [6, 19, 20].

In addition to the above approaches, Bayesian linear regression (BLR) is also used to learn unknown system dynamics, despite its simple linear nature [9]. In [9], weighted Bayesian linear regression (wBLR), which is an extension of BLR is used to estimate unknown dynamics caused by factors such as payload, terrain, or tyre pressure on a ground vehicle while performing repetitive path following tasks. The computational cost is comparatively lesser in wBLR. Hence, this method can quickly and reliably adapt to new situations through fast learning [9]. wBLR approach requires more prior knowledge to identify a linear set of unknown parameters through transformations. In contrast, NN offers a more flexible mechanism for defining the class of nonlinear functions to be used for training a prediction model.

Among these methods, NN and GPR are considered to be more suitable for nonlinear function approximation than wBLR, also with more proven results in literature [4,6,8]. This thesis first carries out a comparative study to select a method from GPR and ANN by evaluating them on a same dataset. The selected approach which has a lesser model error is then further analyzed in the task of learning unknown dynamics of a quadrotor. A detailed description of these two approaches are presented in Chapter 2.

## 1.5 Problem Statement

Basic quadcopter dynamics are of low fidelity due to complex aerodynamic effects and imperfections in hardware. These unknown components result in errors in force and torque models shown in (1.1) and (1.2). Although these errors can be tolerated from a robust controller, the performance will be limited. If these components can be modelled with sufficient accuracy, the flight performance can be increased. Since the

performance of the model-based controllers depends on the model accuracy, capturing a detailed dynamic model is beneficial for the controller algorithm and navigation system design of quadcopters. Using traditional analytical methods to capture these unknown dynamics can be difficult due to the complexity and unknown sources. However, predictable components of these can be estimated through ML techniques.

## 1.6 Objectives and Expected Contributions of the Research

The objectives of this research and the associated contributions are identified as follows.

**Objective 1** Evaluate GPR and ANN for estimating unknown dynamics of a quadcopter.

- Detailed comparison of GPR and ANN to learn unmodelled forces of a quadcopter.
- Performance comparison of different ANN architectures using model error.

**Objective 2** Model the unknown force and torque dynamics using ANN.

- Extending the ANN approach to estimate the unmodelled torque components.

**Objective 3** Demonstrate the improvement of trajectory tracking performance when the unknown dynamic model is incorporated in the controller.

## 1.7 Organization of the Thesis

**Chapter 1** presents the motivation for this work followed by objectives and contributions of this research and the expected outcome.

**Chapter 2** describes the theoretical background of the selected ML techniques and a literature review of the applications of these methods to learn unknown dynamics of a quadrotor UAV.

**Chapter 3** compares GPR and ANN for learning unknown quadcopter dynamics on a dataset captured by manually flying a quadrotor and selects most suitable approach based on the comparison results.

**Chapter 4** includes the detailed methodology of learning unknown quadcopter dynamics using ANN. The unmodelled force and torque are learnt and the model error after adding the learnt term in the dynamics is compared with the nominal model error.

**Chapter 5** demonstrates the results on a simulation of a quadrotor flight controller with a trajectory tracking problem. The results are shown in comparison with the nominal model in the controller.

**Chapter 6** includes the conclusion of this research and the future direction is presented.

# Chapter 2

# Background on Selected ML Techniques

In this chapter, two state-of-the-art ML techniques; GPR and ANN are reviewed. According to the literature, these two techniques have been successfully used in learning unmodelled dynamics of non-linear systems [6, 8, 15, 21].

## 2.1 Artificial Neural Networks (ANN)

### 2.1.1 Theoretical Background of ANN

The concept of NN is developed to mimic the neurons and connections in the human brain, considering the human's ability to learn fast and process complex information. Human brain processes signals via axons between two neurons, so the same concept is adapted in ANNs by having neurons and connections between them. Even though the human brain's ability to learn complex and highly non-linear information is far greater

than today's computing capabilities, for certain complex applications such as image processing, ANNs are approaching comparable performance to that of humans [22].

The architecture of an ANN begins with an input layer followed by one or more hidden layers and finally an output layer, as illustrated in Figure 2.1. The input layer takes inputs and directs them to the neurons in the first hidden layer. The ANN in Figure 2.1 shows a fully connected NN, that is, every neuron in any layer is connected to every neuron in the successive layer. The inputs for each hidden layer are subjected to corresponding weights matrix and an array of biases and the output of each neuron in the hidden layers passes through an activation function to introduce non-linearities to the outputs. When a NN consists of several hidden layers, then the network is called a deep neural network (DNN) [4, 23, 24]. As the number of layers and neurons increase, the NN model equation also become complex, allowing the NN to have more flexibility to capture more complex non-linear behaviours.

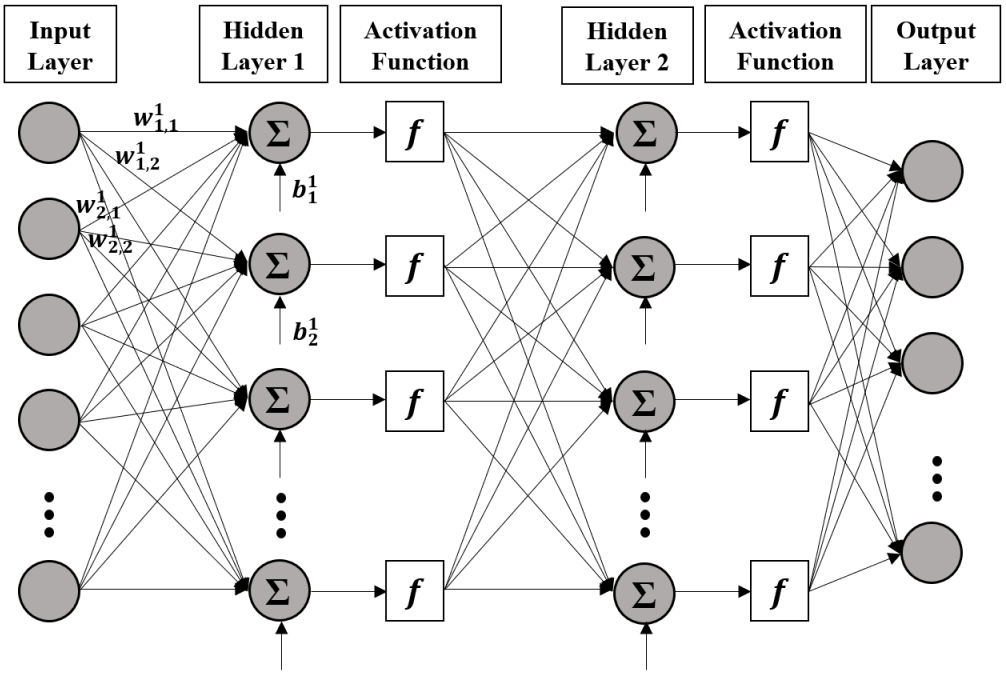



Figure 2.1: General NN architecture with two hidden layers

The flow of an ANN is as follows. Let $\boldsymbol{x}_0$ be the input vector, $\boldsymbol{x}_i$ be the neuron vector in the $i^{th}$ layer, $\boldsymbol{W}_i$ be the weight matrix for the $i^{th}$ layer and $\boldsymbol{b}_i$ be the bias vector for the $i^{th}$ layer. Assuming number of neurons in the $i^{th}$ layer is $n_i$, $\boldsymbol{x}_i$, $\boldsymbol{W}_i$ and $\boldsymbol{b}_i$ are defined as,

$$\boldsymbol{x}_i = [x_1^i, x_2^i, ..., x_n^i]^T \tag{2.1a}$$

$$\boldsymbol{W}_i = \begin{bmatrix} w_{1,1}^i & w_{2,1}^i & ... & w_{n_{i-1},1}^i \\ w_{1,2}^i & w_{2,2}^i & ... & w_{n_{i-1},2}^i \\ \vdots & \vdots & \ddots & \vdots \\ w_{1,n_i}^i & w_{2,n_i}^i & ... & w_{n_{i-1},n_i}^i \end{bmatrix} \tag{2.1b}$$

$$\boldsymbol{b}_i = [b_1^i, b_2^i, ..., b_{n_i}^i]^T \tag{2.1c}$$

Each subsequent neuron vector is calculated by multiplying the previous neuron vector with the corresponding weight matrix and then adding the corresponding bias vector.

$$\boldsymbol{x}_i = \boldsymbol{W}_i \boldsymbol{x}_{i-1} + \boldsymbol{b}_i \tag{2.2}$$

Since this is linear equation, each subsequent layer is a linear function of the previous layer. To include non-linearities, this resultant $\boldsymbol{x}_i$ vector is passed through an activation function. Then, (2.2) can be re-written as,

$$\boldsymbol{x}_i = f_i(\boldsymbol{W}_i \boldsymbol{x}_{i-1} + \boldsymbol{b}_i) \tag{2.3}$$

where $f_i$ denotes the activation function for the $i_{th}$ layer. For a simple ANN with two hidden layers and the output vector ($\boldsymbol{y}$), the layer equations can be combined as in

(2.4).

$$\boldsymbol{y} = f_3\left(\boldsymbol{W}_3\left[f_2\left(\boldsymbol{W}_2\left[f_1(\boldsymbol{W}_1\boldsymbol{x}_0 + \boldsymbol{b}_1)\right] + \boldsymbol{b}_2\right)\right] + \boldsymbol{b}_3\right) \tag{2.4}$$

The activation function can be selected from number of options. Commonly used functions are shown in table 2.1.

Table 2.1: NN Activation Functions

| Name | Equation |
|---|---|
| Linear | $f(x) = x$ |
| Sigmoid | $f(x) = \frac{1}{1+e^{-x}}$ |
| Rectified Linear Unit (ReLU) | $f(x) = max(0, x)$ |
| Hyperbolic Tangent ($tanh$) | $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ |

When the inputs of the training set are associated with corresponding labels or targets, the process is called supervised learning [23]. In supervised learning, the weights and biases are trained using a set of labeled data. This labeled dataset is called the training set. Objective of the training process is to minimize the error between desired output and the network output. After each iteration, the error between desired and trained output is calculated using a separate dataset. This dataset is called the validation set. Due to over-fitting, after certain number of iterations, this validation error will start to increase. The training process will be terminated at this point. The set of weights and biases for every layer when the error is minimum is chosen as the optimal weights and biases. Finally, the trained network is then verified using a testing set independent from the training and validation sets.

### 2.1.2 Current Work on ANN for Learning Unknown Quadcopter Dynamics

ANN is a commonly used ML technique to learn unknown quadcopter dynamics, because of its high flexibility and relatively modest computational cost. This section presents a literature review on the previous work carried out by different researchers on deploying ANN to learn the unmodelled dynamics of a physical system. Since the objectives of this thesis are focused towards the dynamics of a quadcopter, more attention is given to the literature related to quadcopter dynamics.

The main reference for the quadcopter dynamics in this research can be found in [8], which learns the complex aerodynamic effects happened when landing and taking-off a quadrotor UAV. They have stated that unknown aerodynamic effects can be captured as force and torque components. Their dynamic equations are given in (2.5). The states in their system are, position $\boldsymbol{p}$ of the quadcopter expressed in world frame, velocity $\boldsymbol{v}$ of the body frame w.r.t. world frame expressed in world frame, rotation matrix $\boldsymbol{R}$ from body frame to world frame, and the body angular velocity $\boldsymbol{\omega}$ w.r.t. world frame expressed in body frame.

$$\dot{\boldsymbol{p}} = \boldsymbol{v} \tag{2.5a}$$

$$m\dot{\boldsymbol{v}} = m\boldsymbol{g} + \boldsymbol{R}\boldsymbol{f}_u + \boldsymbol{f}_a \tag{2.5b}$$

$$\dot{\boldsymbol{R}} = \boldsymbol{R}S(\boldsymbol{\omega}) \tag{2.5c}$$

$$\boldsymbol{J}\dot{\boldsymbol{\omega}} = \boldsymbol{J}\boldsymbol{\omega} \times \boldsymbol{\omega} + \boldsymbol{\tau}_u + \boldsymbol{\tau}_a \tag{2.5d}$$

Here, $m$ and $\boldsymbol{J}$ are the mass and inertia tensor of the quadcopter, $\boldsymbol{g}$ is the gravity

vector. $\boldsymbol{f}_u$ and $\boldsymbol{\tau}_u$ are the thrust and torque of the quadrotor, expressed in body frame. $S(\boldsymbol{\omega})$ is the skew-symmetric matrix defined in (2.6), where $\omega_x$, $\omega_y$ and $\omega_z$ are angular velocities around $x$, $y$ and $z$ axes.

$$S(\boldsymbol{\omega}) = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}, \tag{2.6}$$

They have considered modelling only the disturbance force $\boldsymbol{f}_a$ to improve their controller. The disturbance torque $\boldsymbol{\tau}_a$ is limited in the landing and take-off application [8]. Their ANN consists of four fully-connected hidden layers activated by $ReLU$ (Rectified Linear Units) function (refer to table 2.1). $ReLU$ activation function is selected because of the faster convergence and higher robustness compared to *sigmoid* functions [8]. They have chosen their inputs as global height $z$, global velocity $\boldsymbol{v}$, attitude rotation matrix $\boldsymbol{R}$ and control input $\boldsymbol{u}$, and output is disturbance force, $\boldsymbol{f}_a$. They have used spectral normalization to guarantee the Lipschitz constant of the ANN, i.e. the maximum ratio between output variations and input variations in the function [25]. The training is done offline in this work, then the result is applied to the dynamic model in the on-board controller in real-time. They have proven results for smoother take-off and landing tasks when the learnt unknown dynamics are included.

The work presented in [18] is also focused on learning unknown dynamics of a quadcopter. Their dynamics are derived using the states $\boldsymbol{s} = [\boldsymbol{p}, \boldsymbol{v}, \boldsymbol{\zeta}, \boldsymbol{\omega}]^T$, where $\boldsymbol{p}$ is the position of the body frame expressed in world frame, $\boldsymbol{v}$ is the velocity of the body frame w.r.t. world frame expressed in world frame, $\boldsymbol{\zeta} = (\phi, \theta, \psi)$ is the Euler angles of the body frame w.r.t. world frame, and $\boldsymbol{\omega}$ is the body angular velocity w.r.t. world frame, expressed in body frame. The dynamic equation is given in (2.7a).

$$\dot{\boldsymbol{s}} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{f}_v \\ \hat{\boldsymbol{R}}\boldsymbol{\omega} \\ \boldsymbol{f}_\omega \end{bmatrix}, \tag{2.7a}$$

where,

$$\hat{\boldsymbol{R}} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \dfrac{\sin\phi}{\cos\theta} & \dfrac{\cos\phi}{\cos\theta} \end{bmatrix}. \tag{2.7b}$$

Here, the unknown linear and angular accelerations, $\boldsymbol{f}_v$ and $\boldsymbol{f}_\omega$ are separately trained using a single hidden layer NN with ReLU activation function, minimizing mean squared prediction error. Input vector of the NN used for learning $\boldsymbol{f}_v$ is $[\boldsymbol{v}, \boldsymbol{\omega}, \sin(\zeta), \cos(\zeta), u_1]$, while the NN which learnt $\boldsymbol{f}_\omega$ used $[\boldsymbol{v}, \boldsymbol{\omega}, \sin(\zeta), \cos(\zeta), u_2, u_3, u_4]$, where $u_1$ is the body thrust in z-direction and $u_2$, $u_3$ and $u_4$ are the body torques in $x$, $y$ and $z$ directions respectively. These control inputs, i.e., $u_1$, $u_2$, $u_3$ and $u_4$ are expressed in body frame. After learning process is completed offline, they have developed a Linear Quadratic Regulator (LQR) controller for a trajectory tracking problem. They have achieved better results when using the learnt unknown model in the controller.

The work in [18] is motivated by [21], which learns unknown dynamics of a helicopter. The dynamic model used in [21] is given in (2.8).

$$\begin{bmatrix} \dot{\boldsymbol{r}} \\ \dot{\boldsymbol{q}} \\ \dot{\boldsymbol{v}} \\ \dot{\boldsymbol{\omega}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{C}_{12}\boldsymbol{v} \\ \frac{1}{2}\boldsymbol{\omega}\boldsymbol{q} \\ \boldsymbol{C}_{12}^T\boldsymbol{g} - \boldsymbol{\omega} \times \boldsymbol{v} + \boldsymbol{f}_v \\ \boldsymbol{f}_\omega \end{bmatrix}, \tag{2.8}$$

where $\boldsymbol{g}$ is the gravity vector and $\boldsymbol{C}_{12}$ is the rotation matrix from body frame to world frame.

The states of the dynamic model in this work are; position of the helicopter $\boldsymbol{r}$ w.r.t. world frame expressed in world frame, orientation of the helicopter $\boldsymbol{q}$ w.r.t. world frame, linear velocity of the helicopter $\boldsymbol{v}$ w.r.t. world frame expressed in body frame and angular velocity of the helicopter $\boldsymbol{\omega}$ w.r.t. world frame expressed in body frame. They have also trained the unknown linear and angular acceleration components, $\boldsymbol{f}_v$ and $\boldsymbol{f}_\omega$ using a DNN as a combination of a quadratic lag model and a two hidden layer NN with ReLU activation.

The MPC controller presented in [1] has also used NN to learn the dynamics of a cart-pole, quadcopter, and an AutoRally vehicle. The states in this work are the configuration $\boldsymbol{q}$ and it's first derivative $\dot{\boldsymbol{q}}$ w.r.t. time. In case of a quadcopter, its configuration can be considered as the position and velocity. The discretized system equation is written as;

$$\boldsymbol{x}_{t+1} = \begin{bmatrix} \boldsymbol{q}_t + \dot{\boldsymbol{q}}_t \Delta t \\ \dot{\boldsymbol{q}}_t + f(\boldsymbol{x}_t, \boldsymbol{u}_t)\Delta t \end{bmatrix}, \tag{2.9}$$

where $\Delta t$ is the discrete time step.

Their task was to learn the unknown term $f$ in the above model. This function $f$ is the second derivative of $\boldsymbol{q}$, i.e., the acceleration. They have used twelve states in the NN model. A fully connected two hidden layer ANN with $TanH$ activation (refer table 2.1) is adopted in this work, comparing the results with three different combinations of hidden-layer neurons for the quadcopter performance. These configurations are in table 2.2. Since their dynamic model has twelve states and four control inputs, they incorporated bootstrapping in their NN model. Bootstrapping provides statistical in-

formation on the statistical distribution of the NN outputs [26]. They have mentioned that the quadrotor resulted repetitive failures without bootstrapping. After learning the model, they have simulated a trajectory tracking task on the quadcopter. According to their simulation results, all three network configurations have shown similar results, but the first and third networks took an extra iteration to achieve their task than the second network with 32 neurons in the hidden layers. Their experiments are only carried out for the AutoRally vehicle.

Table 2.2: NN configurations in [1]

| | Layer 1 neurons | Layer 2 neurons | Activation function |
|---|---|---|---|
| Configuration I | 16 | 16 | *tanh* |
| Configuration II | 32 | 32 | *tanh* |
| Configuration III | 64 | 64 | *tanh* |

A MOdular DEep Recurrent Neural Network (MODERNN) is used in [24] to learn the higher order non-linear dynamics in a quadcopter model. Their algorithm is a novel approach to ANN, which combines different types of NN in one model. They have focused on a trajectory tracking problem, therefore their input to the NN is the sum of motor velocities while the output is the position of the quadrotor. The training process is carried out as an offline batch learning approach to achieve a higher accuracy of the model. In order to optimize a sum of squared errors (SSE) cost function, they have adapted the Levenberg-Marquardt (LM) algorithm [27], which is a gradient descent training method. A simulated quadcopter trajectory tacking algorithm is used in this work to test their algorithm along with a comparison with other available approaches.

Work is presented in [5] does not directly train the unknown dynamics, but NN is used as an identifier as well as in the controller. They have used a single hidden layer NN with sigmoid activation. Their results were validated using a quadcopter simulator,

with and without a payload.

The above literature shows the application of ANN to learn unknown terms in various dynamic models. A variety of ANN architectures presented in these literature exhibits how the number of hidden layers and neurons affected their applications. Motivated by these work, the work presented in this thesis also uses ANN to learn the unknown dynamics of a quadcopter model.

## 2.2 Gaussian Process Regression (GPR)

### 2.2.1 Theoretical Background of GPR

Gaussian process regression is a non-parametric regression method that trains a model as a probabilistic function. According to [28], a Gaussian process (GP) is defined using a set of random variables, where any finite number of them have a joint Gaussian distribution. Its mathematical model is defined by introducing a fixed mean function $(m(x))$ along with a kernel function $(k(x, x'))$ [28]. The kernel $(k(x, x'))$ defines the covariance between two variables: $x$ and $x'$. The mathematical model of a GP is given as,

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')) \tag{2.10}$$

where,

$$m(x) = \mathbb{E}[f(x)] \quad \text{and}$$
$$k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))].$$

Regarding the complexity of having a fixed mean for the whole model in the practical applications, an additional term, which is called the basis function was inserted into this model [28]. There can be more than one basis function in the model. The coefficients of basis functions $\beta$ are estimated during the regression process. By adding the basis functions $h(x)$ to the model, the residuals are estimated as a zero-mean Gaussian process,

$$g(x) = f(x) + h(x)^T\beta \tag{2.11}$$

where $\quad f(x) \sim \mathcal{GP}(0, k(x, x'))$.

The kernel function is defined using a set of hyperparameters. For an example, the squared exponential kernel in (2.14) uses measurement noise, $\sigma_\omega^2$, process variance, $\sigma_\eta^2$, and length scales, $l$ as hyperparameters. These hyperparameters are learnt during the training process. Once the Gaussian process model is learnt, it predicts the corresponding output for a given arbitrary input. For a new input vector $x_*$, the joint distribution between observed function values $y$ and predicted response $f_*$ can be expressed as the probabilistic model in (2.12).

$$\begin{bmatrix} y \\ f_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} k(x,x) + \sigma_n^2 I & k(x, x_*) \\ k(x_*, x) & k(x_*, x_*) \end{bmatrix}\right) \tag{2.12}$$

### 2.2.2 Current Work on GPR for Learning Unknown Quadcopter Dynamics

Several research studies have selected GPR as their learning approach to learn unknown quadcopter dynamics. In [6], a learning-based control process using Gaussian processes is described. The states $\boldsymbol{x}$ used in their paper are, quadcopter's position expressed in world frame $[x, y, z]$, velocity of body frame w.r.t. world frame expressed

in world frame $[\dot{x}, \dot{y}, \dot{z}]$, Euler angles of the body frame in $z$, $y$ and $x$ directions $[\psi, \theta, \phi]$ and angular velocities of the quadcopter w.r.t. world frame expressed in world frame $[\omega_x, \omega_y, \omega_z]$. The control inputs $\boldsymbol{u}$ are desired roll $\phi_{des}$, desired pitch $\theta_{des}$, desired angular velocity around $z$-axis of the body frame $\omega_{z,des}$ and desired velocity in $z$-direction of the world frame $\dot{z}_{des}$. They have controlled the $x$ and $y$ positions of the quadrotor by maintaining constant $z$-position and yaw using a separate controller.

Their dynamics are defined as a combination of two parts; a known model $f(\boldsymbol{x}_k, \boldsymbol{u}_k)$ and an unknown model $g(\boldsymbol{x}_k, \boldsymbol{u}_k)$, as in (2.13). The known model was derived from dynamic equations using first principles, while the unknown model, which is to be learnt will capture the unmodelled dynamics in $f(\boldsymbol{x}_k, \boldsymbol{u}_k)$.

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k) + g(\boldsymbol{x}_k, \boldsymbol{u}_k), \tag{2.13}$$

where $k$ denotes the $k^{th}$ time step.

In [6], the authors have modelled $g(\boldsymbol{x}, \boldsymbol{u})$ as a zero-mean GP. As mentioned in section 1.3, the unknown dynamics are assumed to be non-linear, but in this work, the dynamics are assumed to be linear around a specific operating point with corresponding steady-state input. Given that any kernel with a continuous first derivative can be used for GPR [6], they have selected a squared-exponential kernel having separate length scale for each predictor in the input matrix as in (2.14).

$$k(\boldsymbol{a}_i, \boldsymbol{a}_j) = \boldsymbol{\sigma}_\eta^2 exp\Big( -\frac{1}{2}(\boldsymbol{a}_i - \boldsymbol{a}_j)^T \boldsymbol{M}^{-2}(\boldsymbol{a}_i - \boldsymbol{a}_j)\Big) + \delta_{ij}\boldsymbol{\sigma}_\omega^2 \tag{2.14}$$

where, $\delta_{ij} = 1$ if $i = j$ and 0 otherwise.

This Kernel is parameterized by hyperparameters; measurement noise, $\boldsymbol{\sigma}_\omega^2$, process

variance, $\boldsymbol{\sigma}_{\eta}^2$, and length scales, $\boldsymbol{l}$. The length scales are the diagonal elements of the diagonal matrix $\boldsymbol{M}$. Each predictor has its corresponding length scale, which can be used in the predictions. These hyperparameters were initialized using the standard deviations of the available dataset and estimated while training.

In order to define the GP for an application, these hyperparameters in the kernel should be learnt. In this work, they have used a dataset to learn these parameters by solving a maximum loglikelihood problem using gradient ascent methods.

Once the learning process is completed, $N$ number of past observations $\mathcal{D}$ as given in (2.15) are used to predict the output $g(\boldsymbol{a}^*)$ for an arbitrary input, $a^*$. These past observations are assumed to be noisy, therefore, $\hat{g}(\boldsymbol{a}) = g(\boldsymbol{a}) + \omega$ with $\omega \sim \mathcal{N}(0, \sigma_{\omega}^2)$ where, $g(\boldsymbol{a})$ is the true function value. The joint probability between $\hat{g}$ and $g(\boldsymbol{a}^*)$ is shown in the following equation, which is used to calculate the unknown dynamics for a given input.

$$\mathcal{D} = \{\boldsymbol{a}_i, \hat{g}(\boldsymbol{a}_i)\}_{i=1}^{N} \tag{2.15}$$

$$\begin{bmatrix} \hat{g} \\ g(\boldsymbol{a}^*) \end{bmatrix} \sim \mathcal{N}\left(0_N, \begin{bmatrix} \bar{\boldsymbol{K}} & k^T(\boldsymbol{a}_*) \\ k(\boldsymbol{a}_*) & k(\boldsymbol{a}_*, \boldsymbol{a}_*) \end{bmatrix}\right) \tag{2.16}$$

The work carried out in [29] uses GPR to approximate a performance measure, $J(\boldsymbol{a})$ of the control objective, where $\boldsymbol{a}$ is the controller input. Their goal is to optimize the controller parameters, but since the learning process of $J$ follows GPR this work is also considered as a good example for our research. Same as the work in [6], authors of [29] also predict the function value $J(\boldsymbol{a}^*)$ at an arbitrary input $\boldsymbol{a}^*$ using $n$ number of past observations. Their mean $\mu_n$ and the variance $\sigma_n^2$ functions at an arbitrary input $\boldsymbol{a}^*$ are given in (2.17).

$$\mu_n(\boldsymbol{a}^*) = \ \mathbf{k}_n(\boldsymbol{a}^*)(\boldsymbol{K}_n + \boldsymbol{I}_n\sigma_\omega^2)^{-1}\hat{J}_n \tag{2.17a}$$

$$\sigma_n^2(\boldsymbol{a}^*) = k(\boldsymbol{a}^*, \boldsymbol{a}^*) - \mathbf{k}_n(\boldsymbol{a}^*)(\boldsymbol{K}_n + \boldsymbol{I}_n\sigma_\omega^2)^{-1}\mathbf{k}_n^T(\boldsymbol{a}^*) \tag{2.17b}$$

where, $\boldsymbol{I}_n$ is the $n \times n$ identity matrix, $\hat{J}_n$ is the noisy observations of $J$ with covariance $\sigma_\omega^2$, $\boldsymbol{K}_n$ is the covariance matrix containing the entries $k(\boldsymbol{a}_i, \boldsymbol{a}_j)$ at $i^{th}$ row and $j^{th}$ column and $\mathbf{k}_n(\boldsymbol{a}^*)$ is defined as,

$$\mathbf{k}_n(\boldsymbol{a}^*) = [k(\boldsymbol{a}^*, \boldsymbol{a}_1), k(\boldsymbol{a}^*, \boldsymbol{a}_2), ..., k(\boldsymbol{a}^*, \boldsymbol{a}_n] \tag{2.18}$$

Matèrn kernel in (2.19) is chosen in this work, instead of the commonly used squared-exponential kernel. The hyperparameters in this kernel include measurement noise $\sigma_\omega^2$, prior variance $\sigma_\eta^2$, and length-scales $\boldsymbol{l}$ (diagonal elements in $\boldsymbol{M}$ corresponding to the rate of change of $J$ w.r.t. $\boldsymbol{a}$.

$$k(\boldsymbol{a}_i, \boldsymbol{a}_j) = \sigma_\eta^2(1 + \sqrt{3}r(\boldsymbol{a}_i, \boldsymbol{a}_j))exp(-\sqrt{3}r(\boldsymbol{a}_i, \boldsymbol{a}_j)) \tag{2.19a}$$

where,

$$r(\boldsymbol{a}_i, \boldsymbol{a}_j) = \sqrt{(\boldsymbol{a}_i - \boldsymbol{a}_j)^T\boldsymbol{M}^{-2}(\boldsymbol{a}_i - \boldsymbol{a}_j)} \tag{2.19b}$$

After learning $J$, a safe control algorithm using Bayesian optimization is used to optimize $J$ within a safety margin. Bayesian optimization an approach to find the global maximum of a function expressed as a GP. Since this work is to introduce the control algorithm, only what is relevant for our research is discussed here.

Another research performed to learn unknown quadcopter dynamics using GPR can be found in [19]. Their quadcopter dynamics are formulated from the position $\boldsymbol{r} =$

$[x, y, z]^T$ and the orientation Euler angles $\phi, \theta, \psi$ as;

$$\ddot{\boldsymbol{r}} = \boldsymbol{g}\boldsymbol{z}_w + \frac{1}{m}\boldsymbol{R}\boldsymbol{z}_w\boldsymbol{f}_z \tag{2.20a}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \omega \tag{2.20b}$$

where, $\boldsymbol{z}_w = [0, 0, 1]^T$, $m$, $\boldsymbol{g}$, $\boldsymbol{R}$ and $\boldsymbol{f}$ are the mass of the quadcopter, gravitational acceleration, rotation matrix from body frame to world frame, and the thrust in the body frame expressed in body frame, respectively. The angular velocities of the quadrotor w.r.t. world frame expressed in body frame, $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z]^T$ are the control inputs to the quadcopter. Since the disturbances are not captured in the nominal model, an unknown term is added to each dimension of the above model. The inputs to the GP model are selected as, $\boldsymbol{q} = [r^T, \dot{r}^T, \theta, \phi, \psi]^T$.

$$\ddot{\boldsymbol{r}} = \boldsymbol{g}\boldsymbol{z}_w + \frac{1}{m}\boldsymbol{R}\boldsymbol{z}_w\boldsymbol{f}_z + \begin{bmatrix} \mathcal{GP}_1(0, k(\boldsymbol{q}, \boldsymbol{q}')) \\ \mathcal{GP}_2(0, k(\boldsymbol{q}, \boldsymbol{q}')) \\ \mathcal{GP}_3(0, k(\boldsymbol{q}, \boldsymbol{q}')) \end{bmatrix} \tag{2.21a}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \boldsymbol{\omega} + \begin{bmatrix} \mathcal{GP}_4(0, k(\boldsymbol{q}, \boldsymbol{q}')) \\ \mathcal{GP}_5(0, k(\boldsymbol{q}, \boldsymbol{q}')) \\ \mathcal{GP}_6(0, k(\boldsymbol{q}, \boldsymbol{q}')) \end{bmatrix} \tag{2.21b}$$

Mean $m$ and the variance $\sigma^2$ at a query point $x_*$ are expressed as;

$$m(x_*) = k_*^T(\boldsymbol{K} + \sigma_n^2\boldsymbol{I})^{-1}y_w \tag{2.22a}$$

$$\sigma^2(x_*) = k(x_*, x_*) - k_*^T(\boldsymbol{K} + \sigma_n^2\boldsymbol{I})^{-1}k_* \tag{2.22b}$$

Here, $\boldsymbol{I}$ is the corresponding identity matrix, $y_w$ is a collection of $w$ number of measurements, $\boldsymbol{K}$ is the covariance matrix containing the entries $k(x_i, x_j)$ at $i^{th}$ row and $j^{th}$ column and $k_*$ is defined similar to (2.23) as,

$$k_* = [k(x_1, x_*), k(x_2, x_*), ..., k(x_w, x_*)] \tag{2.23}$$

Since the computation of the inverse of the kernel takes a significantly higher amount of time which causes practical deficiencies, they have introduced a method to speed-up the process. When the quadrotor is moving forward, their kernel is updated by adding new points and removing old and unnecessary points. This way, the kernel can be defined smaller, without using the whole dataset.

A simulation of trajectory tracking problem for a quadcopter is carried out to validate the results in [19]. A disturbance wind model is added to the system and the unknown dynamics are learnt online during the flight. Their results show a significant reduction in the trajectory error once the unknown dynamics learnt by GPR is added to the system model.

[20] presents another approach learning unknown quadcopter dynamics using GPR. They have proposed a general dynamic model as in (2.24), as a combination of known dynamic functions $h$ and $g$ with an unknown disturbance term $d$, where $\boldsymbol{x}$ is the state of the system and $\boldsymbol{u}$ is the control input. The unknown term $d$ is a deterministic, state dependant and assumed to be locally Lipschitz continuous.

$$f(\boldsymbol{x}, \boldsymbol{u}, d) = h(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u} + d \tag{2.24}$$

This disturbance model $d$ is then estimated through GPR. As observations in the learning process, they have calculated the disturbance model prediction $\hat{d}$ using (2.25). An approximation of state derivatives $\hat{f}$, was obtained from numerical differentiation.

$$\hat{d}(\boldsymbol{x}) = \hat{f}(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{x})) - h(\boldsymbol{x}) - g(\boldsymbol{x})\boldsymbol{u}(\boldsymbol{x}) \tag{2.25}$$

This $\hat{d}$ is assumed to be a noisy measurement of $d$, a zero-mean Gaussian noise with variance $\sigma_n^2$. After learning this unmodelled disturbance as a GP with squared exponential kernel $\boldsymbol{K}$, the prediction of the function $\bar{d}$ at a given input $\boldsymbol{x}_*$ can be expressed as;

$$\bar{d}(\boldsymbol{x}_*) = \mu(\boldsymbol{x}_*) + \boldsymbol{K}(\boldsymbol{x}_*, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n \boldsymbol{I})^{-1}(\hat{d}(\boldsymbol{x}) - \mu(\boldsymbol{X})) \tag{2.26}$$

where, $\mu(\boldsymbol{X})$ is the mean function at the past inputs $\boldsymbol{X} = [x_1, x_2, ..., x_N]$.

A disturbance set $\hat{\mathcal{D}}$ is calculated from these predictions as follows.

$$\hat{\mathcal{D}}(\boldsymbol{x}) = [\bar{d}(\boldsymbol{x}_*) - m\sigma_*(\boldsymbol{x}), \bar{d}(\boldsymbol{x}_*) + m\sigma_*(\boldsymbol{x})]; \quad \forall m \in \mathbb{R}^+ \tag{2.27}$$

This set $\hat{\mathcal{D}}$ is then used to perform a reachability analysis followed by an optimal safe control law, which is the main target in their work. Their results are tested in a quadrotor altitude controller, which increased the performance by adding this disturbance model to the dynamics. The state-space model for the quadcopter system is presented in (2.28), comprised of the states; altitude of the quadrotor $x_1$, vertical velocity $x_2$ and average angular velocity of the four rotors $x_3$.

$$\dot{x_1} = x_2 \tag{2.28a}$$

$$\dot{x_2} = k_T x_3^2 + g + d(x) \tag{2.28b}$$

$$\dot{x_3} = k_p(u - x_3) \tag{2.28c}$$

where, $u$ is the control input taken as the desired average angular velocity, $k_p$ is a time constant defined as $1/k_p = 0.15s$, $g = -9.81ms^{-2}$ is the gravitational acceleration and $d$ is the unknown disturbance model, predicted by GPR.

## 2.3 Model-Based Control of Quadcopters

Adding unknown dynamic components to the nominal dynamic model can improve the performance of the model-based control of the quadrotor UAVs [6, 8, 18]. The application of the learnt unknown dynamics in the controller can be found in many literatures, including the papers presented in sections 2.1.2 and 2.2.2. This section includes a discussion about currently available model-based controllers with learnt unknown dynamic models.

Linear quadratic regulator (LQR) is one of the common model-based controllers found in literature [18]. Although LQRs are designed for linear systems, it can be implemented on non-linear systems once the dynamics are linearized. The work in [18] has developed an LQR controller for trajectory tracking. The linearized dynamics of their system is expressed as,

$$\boldsymbol{s}_{n+1} = \boldsymbol{A}\boldsymbol{s}_n + \boldsymbol{B}\boldsymbol{u}_n, \tag{2.29}$$

where $n$ is the $n^{th}$ time step. The state vector and the control inputs are represented as $s$ and $\boldsymbol{u}$, respectively. Vector $\boldsymbol{s}$ includes the basic dynamic states and the learnt dynamics from a NN. When the desired states and the nominal open-loop control

signal are designated as $\boldsymbol{s}_*$ and $\boldsymbol{u}_*$, the state error $\bar{\boldsymbol{s}}$ and input compensation $\bar{\boldsymbol{u}}$ are defined as,

$$\bar{\boldsymbol{s}}_n = \boldsymbol{s}_n - \boldsymbol{s}_{*n} \tag{2.30a}$$

and

$$\bar{\boldsymbol{u}}_n = \boldsymbol{u}_n - \boldsymbol{u}_{*n}. \tag{2.30b}$$

Once (2.29) is expressed in terms of these errors, the error dynamics become

$$\bar{\boldsymbol{s}}_{n+1} = \boldsymbol{A}\bar{\boldsymbol{s}}_n + \boldsymbol{B}\bar{\boldsymbol{u}}_n. \tag{2.31}$$

These state errors are used to calculate a quadratic cost function $J$ as defined in (2.32). The LQR generated a controller to minimize $J$.

$$J_N(\bar{\boldsymbol{s}}_0) = \sum_{n=0}^{N} \left( \bar{\boldsymbol{s}}_n^T \boldsymbol{Q} \bar{\boldsymbol{s}}_n + \bar{\boldsymbol{u}}_n^T \boldsymbol{R} \bar{\boldsymbol{u}}_n \right) \tag{2.32}$$

Here, $N$ is the number of data points in the horizon and $\boldsymbol{Q}$ and $\boldsymbol{R}$ are positive definite matrices. The closed-loop control inputs are then calculated from,

$$\boldsymbol{u}_n = \boldsymbol{u}_{*n} + \boldsymbol{K}(\boldsymbol{s}_n - \boldsymbol{s}_{*n}). \tag{2.33}$$

The time-invariant state feedback matrix $\boldsymbol{K}$ should be defined so that the cost $J$ is minimized.

Another control approach to achieve stabilization and trajectory tracking is the Model predictive control (MPC) [1]. The work presented in [1] have improved the classical MPC approach which is capable of blending the learnt dynamic model with the controller. This algorithm is called information theoretic MPC, which is based on

sampling the trajectories.

The discrete-time stochastic dynamic model is defined as,

$$\boldsymbol{x}_{t+1} = \mathbf{F}(\boldsymbol{x}_t, \boldsymbol{u}_t) \tag{2.34}$$

where $\boldsymbol{x}_t$ is the state vector of the system and $\boldsymbol{u}_t$ is the commanded control input at $t^{th}$ time step. The actual input $\boldsymbol{v}$ is assumed to be corrupted by a normally distributed noise and expressed as $v_t \sim \mathcal{N}(\boldsymbol{u}_t, \Sigma)$. The algorithm depends on the hyperparameters, system noise $\Sigma$, terminal cost $\phi$, instantaneous state cost $q$ and a positive scalar variable $\lambda$. For the quadcopter control system, $\lambda = 1$ and $q$ is defined as,

$$q(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{x}_d)^T \boldsymbol{Q} (\boldsymbol{x} - \boldsymbol{x}_d) + 100000C \tag{2.35}$$

where $\boldsymbol{x}_d$ is the desired state and $\boldsymbol{Q}$ is a diagonal matrix. The term $100000C$ represents $\phi$, where $C$ is triggered if the quadcopter is crashed.

They have introduced a sampling approach called importance sampling. The MPC algorithm generates a control sequence and the first element of the control sequence is executed. In the next iteration, the remaining elements of this sequence are taken as the importance sampling trajectory. The minimum cost $\beta$ is subtracted from the cost function to guarantee that at least one trajectory has low cost. The cost at $k^{th}$ sample is,

$$S(\boldsymbol{\varepsilon}^k) = \sum_{t=0}^{T} \left( q(\boldsymbol{x}_t) + \lambda \boldsymbol{u}_{t-1}^T \Sigma^{-1} \epsilon_{t-1}^k \right) \tag{2.36}$$

where $\boldsymbol{\varepsilon}^k = \{\epsilon_0^k, \epsilon_1^k, ..., \epsilon_{T-1}^k\}$ is the noise sequence throughout the time steps at the

$k^{th}$ sample. At $k$, an importance sampling weight $w(\boldsymbol{\varepsilon}^k)$ is defined as

$$w(\boldsymbol{\varepsilon}^k) = \frac{1}{\eta} exp\Big( -\frac{1}{\lambda}(S(\boldsymbol{\varepsilon}^k) - \beta)\Big). \tag{2.37}$$

$\eta$ is approximated using Monte-Carlo estimate es follows.

$$\eta = \sum_{k=0}^{K-1} exp\Big( -\frac{1}{\lambda}(S(\boldsymbol{\varepsilon}^k) - \beta)\Big) \tag{2.38}$$

Finally, the control input for $K$ samples at $t^{th}$ time step is computed by,

$$\boldsymbol{u}_t = \sum_{k=1}^{K} w(\boldsymbol{\varepsilon}^k)\epsilon_t^k. \tag{2.39}$$

The first element $\boldsymbol{u}_0$ is sent to the actuators, while the remainder is used in the next iteration.

A nonlinear feedback linearization controller is formulated in [8] to develop a trajectory tracking controller. The disturbance force learnt from ANN is added to the dynamic model to improve the performance. The trajectory error is defined as

$$\tilde{\boldsymbol{p}} = \boldsymbol{p} - \boldsymbol{p}_d, \tag{2.40}$$

where $\boldsymbol{p}$ and $\boldsymbol{p}_d$ are the observed and desired positions, respectively. This model is transferred into a reference velocity tracking problem by introducing a composite variable $\boldsymbol{s}$,

$$\boldsymbol{s} = \dot{\tilde{\boldsymbol{p}}} + \Lambda\tilde{\boldsymbol{p}} = \dot{\boldsymbol{p}} - \boldsymbol{v}_r \tag{2.41}$$

where $\Lambda$ is a positive definite diagonal matrix. Now the reference velocity becomes

$$\boldsymbol{v}_r = \dot{\boldsymbol{p}}_d - \Lambda\tilde{\boldsymbol{p}}. \tag{2.42}$$

Then an approximation to the learnt disturbance force $\hat{\boldsymbol{f}}_a$ is evolved to compute the desired rotor force $\boldsymbol{f}_d$ as follows.

$$\boldsymbol{f}_d = \bar{\boldsymbol{f}}_d - \hat{\boldsymbol{f}}_a \tag{2.43}$$

Here,

$$\bar{\boldsymbol{f}}_d = m\dot{\boldsymbol{v}}_r - \boldsymbol{K}_v\boldsymbol{s} - m\boldsymbol{g}, \tag{2.44}$$

where $\boldsymbol{K}_v$ is the velocity gain, $m$ is the mass of the quadcopter and $\boldsymbol{g}$ is the gravitational acceleration.

Finally, the control input of the current time step $\boldsymbol{u}_k$ is calculated from the following equation.

$$\boldsymbol{u}_k = \boldsymbol{B}_\circ^{-1}\begin{bmatrix}\left(\bar{\boldsymbol{f}}_d - \hat{\boldsymbol{f}}_a\right).\hat{\boldsymbol{k}} \\ \boldsymbol{\tau}_d\end{bmatrix} \tag{2.45}$$

Here, $\hat{\boldsymbol{k}}$ is the unit vector in the direction of the rotor thrust. Usually, this direction is the $z$ direction in the body reference frame. The matrix $\boldsymbol{B}_\circ$ is the relationship between rotor thrust and torques defined by,

$$\boldsymbol{B}_\circ = \begin{bmatrix} C_T & C_T & C_T & C_T \\ 0 & C_T l_{arm} & 0 & -C_T l_{arm} \\ -C_T l_{arm} & 0 & C_T l_{arm} & 0 \\ -C_Q & C_Q & -C_Q & C_Q \end{bmatrix}, \tag{2.46}$$

where $l_{arm}$ is the rotor arm length and $C_T$ and $C_Q$ are the rotor thrust and torque

coefficients respectively.

The desired torque $\boldsymbol{\tau}_d$ is calculated from (2.47), were, $\boldsymbol{K}_\omega$ is the angular velocity gain, $\boldsymbol{J}$ is the quadrotor's inertia tensor, and $\boldsymbol{\omega}$ is the observed angular velocity. Parameters $\boldsymbol{\omega}_r$ and $\dot{\boldsymbol{\omega}}_r$ are the reference angular velocity defined equivalent to $\boldsymbol{v}_r$ in (2.41) and its first derivative, respectively.

$$\boldsymbol{\tau}_d = \boldsymbol{J}\dot{\boldsymbol{\omega}}_r - \boldsymbol{J}\boldsymbol{\omega} \times \boldsymbol{\omega}_r - \boldsymbol{K}_\omega(\boldsymbol{\omega} - \boldsymbol{\omega}_r) \tag{2.47}$$

# Chapter 3

# Comparison of GPR and ANN for Modelling the Unknown Dynamics

As per the first objective of this research, the most suitable ML approach for our application had to be selected between ANN and GPR. An experiment was carried out for a dataset gathered using AscTec Hummingbird quadrotor UAV to learn the unknown disturbance force in (2.5b). According to the literature, the dominant component in this disturbance model is the aerodynamic drag force [13], mentioned in (1.3). This drag force has a linear relationship with the body velocity ${}^b\dot{\boldsymbol{p}}$. Since it is easier to train a NN for a linear behaviour, (2.5b) is converted into the body fixed reference frame. The terms are defined as in (2.5b) and (1.3).

$$ {}^b\boldsymbol{f}_a = \boldsymbol{R}^T(m\dot{\boldsymbol{v}} - (m\boldsymbol{g} + \boldsymbol{R}\boldsymbol{f}_u)) = -\boldsymbol{K}_d{}^b\dot{\boldsymbol{p}} \tag{3.1} $$

This disturbance force was then learnt using NN and GPR separately, taking the body-frame velocity ${}^b\boldsymbol{v}$ ($={}^b\dot{\boldsymbol{p}}$) as the inputs. Here, the body-frame velocity refers to

the velocity of the quadcopter w.r.t. world frame expressed in body frame. The three components in the output ${}^{b}\boldsymbol{f}_{a}$ were trained as three different processes. Although the behaviour was assumed to be linear, this system has unknown non-linearities as well. Therefore, errors in the learnt model are expected.

## 3.1 Results of Learning Unknown Dynamics with GPR

As the first approach, GPR was used to learn the unknown disturbance force mentioned above. In order to proceed with GPR, a suitable kernel function had to be selected. According to the literature, any kernel function which has a continuous first derivative can be used [6]. Referring to [6], the squared-exponential kernel (refer to (3.2)) was chosen. This thesis followed the same procedure as in the reference paper, with a separate length scale for each predictor (i.e., velocity components in $x$, $y$ and $z$ directions) of the input matrix. Hyperparameters, process variance $\boldsymbol{\sigma}_f$ and length scales $l$ (diagonal elements of the diagonal matrix $\boldsymbol{M}$) were initialized using the standard deviations of the available dataset and they were estimated while training. Table 3.1 includes the initial and estimated hyperparameters.

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{\sigma}_f^2 exp\Big( -\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{x}_j)^T \boldsymbol{M}^{-2}(\boldsymbol{x}_i - \boldsymbol{x}_j)\Big) \tag{3.2}$$

Here, $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are two samples of the dataset.

Since a zero-mean GP is assumed, a basis function was included in the GPR model. A linear basis function in (3.3) was chosen in this case, because the relationship between inputs and outputs are assumed to be linear (refer to (3.1)).

$$h(\boldsymbol{x}_i) = [1, \boldsymbol{x}_i] \tag{3.3}$$

Now, the equation for GP becomes,

$$g(\boldsymbol{x}) = f(\boldsymbol{x}) + h(\boldsymbol{x})\beta \tag{3.4}$$

where, $f(\boldsymbol{x})$ is a zero-mean GP,

$$f(\boldsymbol{x}) \sim \mathcal{GP}(0, k(\boldsymbol{x}, \boldsymbol{x}')) \tag{3.5}$$

The coefficients $\beta$ associated with the basis function are presented in table 3.2.

Table 3.1: Hyperparameters table

| | Hyperparameter | Initialized value | Trained value |
|---|---|---|---|
| $f_x$ | Process variance, $\boldsymbol{\sigma}_f^2$ | 0.1903 | 0.2425 |
| | Length scale 1, $l_1$ (for $v_x$) | 1 | 0.1140 |
| | Length scale 2, $l_2$ (for $v_y$) | 1 | 0.1019 |
| | Length scale 3, $l_3$ (for $v_z$) | 1 | 0.1653 |
| $f_y$ | Process variance, $\boldsymbol{\sigma}_f^2$ | 0.2186 | 0.1770 |
| | Length scale 1, $l_1$ (for $v_x$) | 1 | 0.1794 |
| | Length scale 2, $l_2$ (for $v_y$) | 1 | 0.1073 |
| | Length scale 3, $l_3$ (for $v_z$) | 1 | 0.3064 |
| $f_z$ | Process variance, $\boldsymbol{\sigma}_f^2$ | 1.0342 | 1.1678 |
| | Length scale 1, $l_1$ (for $v_x$) | 1 | 0.1822 |
| | Length scale 2, $l_2$ (for $v_y$) | 1 | 0.0891 |
| | Length scale 3, $l_3$ (for $v_z$) | 1 | 0.2326 |

A subset of 1500 data points from the gathered dataset was selected as the past observations for the training process. The disturbance force ${}^b\boldsymbol{f}_a$ was then learnt as a GP from this training set. The next 1500 data points were then selected to test

Table 3.2: Coefficients of basis function

| | Coefficient | Trained value |
|---|---|---|
| $f_x$ | $\beta_1$ | -0.0199 |
| | $\beta_{x1}$ (for $v_x$) | -0.0762 |
| | $\beta_{x2}$ (for $v_y$) | 0.0099 |
| | $\beta_{x3}$ (for $v_y$) | 0.0123 |
| $f_y$ | $\beta_1$ | -0.0368 |
| | $\beta_{x1}$ (for $v_x$) | -0.0448 |
| | $\beta_{x2}$ (for $v_y$) | -0.1499 |
| | $\beta_{x3}$ (for $v_y$) | -0.0114 |
| $f_z$ | $\beta_1$ | -0.2598 |
| | $\beta_{x1}$ (for $v_x$) | -0.0755 |
| | $\beta_{x2}$ (for $v_y$) | 0.2959 |
| | $\beta_{x3}$ (for $v_y$) | -0.0992 |

the validity of the generated GPR model. The prediction of ${}^b\boldsymbol{f}_a$ for the new input velocities in this testing set (considering each velocity as $\boldsymbol{x}_*$) was calculated from the (2.16).

The results from GPR were compared with the nominal ${}^b\boldsymbol{f}_a$ calculated from the measurements for both training and testing sets, as shown in Figure 3.1. The first 1500 data points corresponds to the training set, while the next 1500 points represents the testing set where the nominal model is plotted as solid lines and the GP prediction is shown in dotted lines.

According to the above plot, the GP prediction for the training set exactly follows the expected values of the nominal model error, but the testing set has a notable error. Since this behaviour yields to a significant decrement in the root-mean-square error (RMSE) if the whole training and testing sets are chosen, only testing set was considered in RMSE calculations. The nominal model error (i.e. nominal ${}^b\boldsymbol{f}_a$) has a RMSE of $0.563N$, while the RMSE in the model error with the GPR model (i.e.

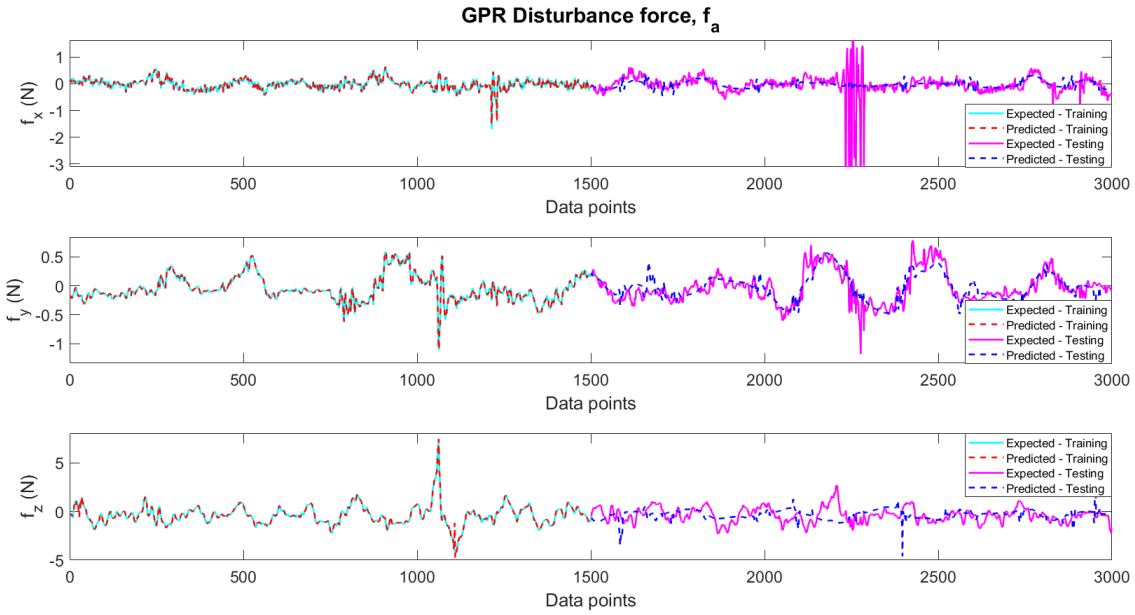



Figure 3.1: Unknown force ${}^{b}\boldsymbol{f}_a$ modelled by GPR

nominal model error + ${}^{b}\boldsymbol{f}_a$ trained by GPR) is $0.6128N$, which is 8.8% higher than the nominal model error.

## 3.2 Results of Learning Unknown Dynamics with ANN

This section includes the results obtained from training ${}^{b}\boldsymbol{f}_a$ using ANN. Similar to the GPR method, the body frame velocities ${}^{b}\boldsymbol{v}$ were selected as inputs to the network, which was designed with one hidden layer having three neurons. Since a linear relationship is assumed between ${}^{b}\boldsymbol{f}_a$ and ${}^{b}\boldsymbol{v}$, a linear activation function was included after the hidden layer. The training and testing sets were also chosen as the same used in GPR. During the training process, the training set was divided among training, validation and testing sets having 70%, 15% and 15% of the data respectively. The

preferred NN architecture in this work is illustrated in Figure 3.2.

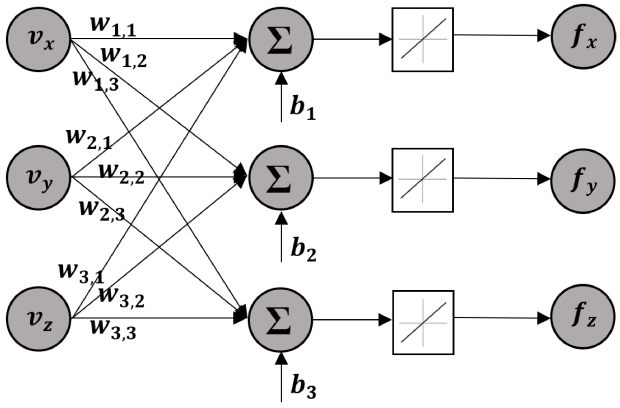



Figure 3.2: Neural Network architecture with one hidden layer and linear activation

According to the above network, the output ${}^b\boldsymbol{f}_a$ can be expressed using the weight matrix ($\boldsymbol{W}$) and bias vector $\boldsymbol{b}$ matrices as follows.

$$ {}^b\boldsymbol{f}_a = \boldsymbol{W}{}^b\boldsymbol{v} + \boldsymbol{b} \tag{3.6} $$

where,

$$ \boldsymbol{W} = \begin{bmatrix} w_{1,1} & w_{2,1} & w_{3,1} \\ w_{1,2} & w_{2,2} & w_{3,2} \\ w_{1,3} & w_{2,3} & w_{3,3} \end{bmatrix} \quad \text{and} \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}. $$

After desired number of iterations, these weight and bias matrices were trained from Bayesian Regularization backpropagation algorithm. This algorithm creates a well-generalized network by adjusting a linear combination of weights and squared errors, while minimizing this linear combination. The trained weights and biases were used to generate (3.6) to predict the function value of ${}^b\boldsymbol{f}_a$ at any value of the input ${}^b\boldsymbol{v}$. The maximum number of iterations was set to be 1000.

After the network was built with trained weights and biases, the predicted output was generated for both training and testing sets. The result obtained for the dataset is included in Figure 3.3. According to the Figure, an error has been occurred between the expected and predicted outputs, in both training and testing sets. This result sums up that the assumption of a linear relationship is occurred and trained, but there are more non-linear relations as well in the model. Comparing the RMSE between the nominal model and trained model, it has been reduced from $0.563N$ to $0.5273N$, which is a reduction of 6.3% from the nominal model.


Figure 3.3: Unknown force ${}^{b}\boldsymbol{f}_a$ modelled by ANN

## 3.3 Performance Comparison of GPR and ANN

In section 3.1 and section 3.2, the disturbance force was modelled using GPR and ANN respectively. With reference to the training set, the prediction of both GPR and ANN followed the nominal model with lesser error and the GPR model showed better

results following the exact model. This behaviour is expected because the model was generated by feeding the same outputs to both GPR and ANN. But, when considering the testing set, which is more important for predictions at unknown inputs, the GPR model had higher errors than the ANN model. This behaviour is projected in Figures 3.1 and 3.3.

In order to have a further agreement on the errors, the RMSE for both GPR model and ANN model were compared in the Figure 3.4. This RMSEs were calculated only for the testing sets because it reflects the practical situations, i.e., the predictions for unknown force model at new inputs that are not used in training. As per this chart, the RMSE has been reduced in ANN predictions from 6.3% than the nominal model, while it is increased from 8.8% in the GPR model.

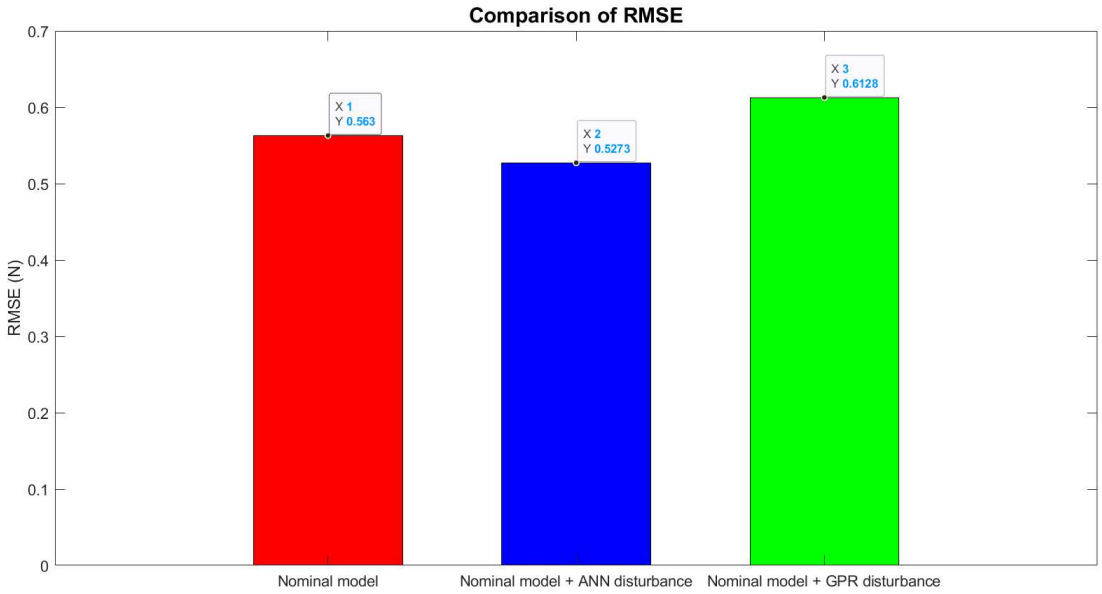



Figure 3.4: RMSE comparison of GPR and ANN w.r.t. the nominal model

The next concern when choosing the better algorithm is the time consumed for the training process. Lesser time consumption is essential for online training to avoid the lags in the controller, which the quadcopter's stability depends on. Comparing the

selected approaches, ANN took only 10.33s, while the GPR spent 87.07s on training.

When the RMSE for the testing set and the time consumption is taken into account, it is apparent that the ANN has more advantage than GPR. Although the performance of GPR was low in our work, GPR has proven successful results in the literature [6, 19, 29]. Our comparison is an initial approach for choosing a better method for learning unknown dynamics of a quadrotor. Over-fitting in the training set is one of the causes for this behaviour. Due to the less flexibility of GPR, this was hard to control. Compared to GPR, ANN has more flexibility to adapt to our dataset and it showed lesser RMSE as well.

Considering the aforementioned advantages, ANN approach was chosen over GPR method for further implementations in this research.

## 3.4 Summary

In this chapter, a comparison of GPR and ANN was presented. The disturbance force ${}^{b}\boldsymbol{f}_{a}$ was learnt using both approaches and the results were compared. The dataset was divided into training and testing sets. The training set was used to learn the model for ${}^{b}\boldsymbol{f}_{a}$ and testing set was used to predict ${}^{b}\boldsymbol{f}_{a}$ for new inputs which were not used in training. According to the results, both approaches could predict the output similar to expected value, since the model was generated using the same training set. However, the predicted output for the testing set had considerable errors in both approaches. The GPR approach had higher errors than ANN method.

In the comparison of RMSEs for the testing set, the RMSE in GPR approach was even higher than the RMSE of the nominal model. ANN approach could reduce the RMSE from $0.563N$ to $0.5273N$ compared to the nominal model. In addition, GPR showed

over-fitting and was not flexible to adapt to the dataset. On the other hand, ANN has more flexibility for our work. Further investigation is recommended for using GPR. Since this is an initial comparison, the better method was chosen from the obtained results. Hence, ANN approach was considered as the most suitable method for our application.

# Chapter 4

# Detailed Evaluation of Learning of Unknown Quadcopter Dynamics Using ANN

In this chapter, application of the selected ML approach, ANN to learn unknown quadcopter dynamics is discussed. First, the complete dynamic model of the quadcopter selected in this work is introduced in section 4.1, then the progressive approach for learning unknown dynamic model using ANN is described in the following sections. The effect of changing the ANN architecture on increasing the complexity and accuracy of the model is also discussed.

## 4.1 Quadcopter Dynamics

The dynamics of the quadcopter is expressed as a combination of a known dynamic model derived from the Newton-Euler equations and an unknown term. This com-

bined dynamic model has the states, position $\boldsymbol{p}$, velocity ${}^{b}\boldsymbol{v}$ of the quadcopter w.r.t world frame, expressed in body frame, rotation matrix ${}^{w}\boldsymbol{R}_{b}$ from body reference frame to world frame and the angular velocity ${}^{b}\boldsymbol{\omega}$ of the body frame w.r.t. world frame, expressed in body frame. The aerodynamic disturbances are the dominant components in the unmodelled dynamics. These aerodynamic disturbances affect the force and torque model of the quadrotor [8]. Hence, unknown dynamic terms are included in the force and torque in the basic dynamic model. In the quadcopter dynamic model defined in (4.1), the unknown terms ${}^{b}\boldsymbol{f}_{a} = [f_x, f_y, f_z]^T$ and ${}^{b}\boldsymbol{\tau}_{a} = [\tau_x, \tau_y, \tau_z]^T$ refer to the unknown force and torque of the body frame w.r.t. the inertial frame, expressed in body frame.

$$\dot{\boldsymbol{p}} = {}^{w}\boldsymbol{R}_{b}\,{}^{b}\boldsymbol{v} \tag{4.1a}$$

$$m\,{}^{b}\boldsymbol{a} = m\,{}^{w}\boldsymbol{R}_{b}^{T}\boldsymbol{g} + {}^{b}\boldsymbol{T} + {}^{b}\boldsymbol{f}_{a} \tag{4.1b}$$

$${}^{w}\dot{\boldsymbol{R}}_{b} = {}^{w}\boldsymbol{R}_{b}[{}^{b}\boldsymbol{\omega}]_{\times} \tag{4.1c}$$

$$\boldsymbol{J}\,{}^{b}\boldsymbol{\alpha} = \boldsymbol{J}\,{}^{b}\boldsymbol{\omega} \times {}^{b}\boldsymbol{\omega} + {}^{b}\boldsymbol{\Gamma} + {}^{b}\boldsymbol{\tau}_{a} \tag{4.1d}$$

The skew-symmetric matrix $[{}^{b}\boldsymbol{\omega}]_{\times}$ is defined as,

$$[{}^{b}\boldsymbol{\omega}]_{\times} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}, \tag{4.2}$$

where ${}^{b}\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z]^T$.

In this dynamic model (see (4.1)), $m$ and $\boldsymbol{J}$ are the mass and the inertia of the quadcopter, $\boldsymbol{g}$ is the gravity vector defined as $\boldsymbol{g} = [0, 0, -9.81]^T$, ${}^{b}\boldsymbol{a}$ and ${}^{b}\boldsymbol{\alpha}$ are

the quadcopter's linear and angular accelerations w.r.t the world frame and ${}^{b}\boldsymbol{T} = [0, 0, T_z]^T$ and ${}^{b}\boldsymbol{\Gamma} = [\Gamma_x, \Gamma_y, \Gamma_z]^T$ are the total thrust and torque of the body frame w.r.t the world frame. The terms ${}^{b}\boldsymbol{a}$, ${}^{b}\boldsymbol{\alpha}$, ${}^{b}\boldsymbol{T}$ and ${}^{b}\boldsymbol{\Gamma}$ are expressed in the body frame.

The body thrust in $z$-direction $T_z$ and the body torques $\Gamma_x$, $\Gamma_y$ and $\Gamma_z$ can be calculated from the velocities of four motors, $u_1$, $u_2$, $u_3$ and $u_4$. This relationship can be expressed as;

$$\begin{bmatrix} T_z \\ \Gamma_x \\ \Gamma_y \\ \Gamma_z \end{bmatrix} = \begin{bmatrix} C_T & C_T & C_T & C_T \\ 0 & C_T l_{arm} & 0 & -C_T l_{arm} \\ -C_T l_{arm} & 0 & C_T l_{arm} & 0 \\ -C_Q & C_Q & -C_Q & C_Q \end{bmatrix} \begin{bmatrix} u_1^2 \\ u_2^2 \\ u_3^2 \\ u_4^2 \end{bmatrix} \tag{4.3}$$

where, $l_{arm}$, is the length from CG to a rotor axis and $C_T$ and $C_Q$ are the thrust and torque coefficients respectively.

## 4.2 Data Selection

In order to model the above nominal dynamics in (4.1), the quadcopter's position, linear and angular velocities and accelerations, the velocities of each individual motor and the quadcopter parameters: mass, inertia, and force and torque coefficients are required. A dataset capturing necessary measurement was gathered by manually flying an AscTec Hummingbird quadcopter under an OptiTrack motion capture system. A segment of the path that the quadcopter was flown is selected for the unknown dynamics learning process. The path that the quadcopter followed, along with the selected subset of that path is shown in Figure 4.1.

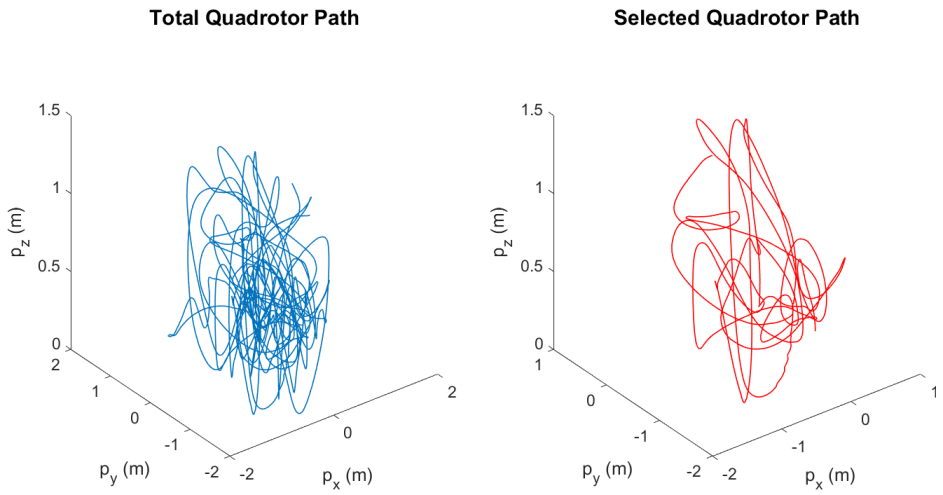



Figure 4.1: Quadrotor path and selected subset for learning

The position data were directly taken from the OptiTrack measurements, while the motor velocities were accessible through the AscTec Hummingbird API (Application programming interface). Although the acceleration measurements were available from the IMU (Inertial measurement unit), those were not used in the calculations since these data are corrupted by a considerable bias and noise. Therefore, the velocity and acceleration were calculated by differentiating the position data captured by OptiTrack. The fixed parameters of the quadcopter were collected from the literature on AscTec Hummingbird quadcopter, [30] presented in Table 4.1.

Table 4.1: AscTec Hummingbird Quadrotor Parameters

| Parameter | Value |
|---|---|
| Mass, $m$ | 0.7 $kg$ |
| Arm length, $l_{arm}$ | 0.17 $m$ |
| Moment of inertia around $x$ axis, $J_{xx}$ | 0.00071 $kgm^2$ |
| Moment of inertia around $y$ axis, $J_{yy}$ | 0.00071 $kgm^2$ |
| Moment of inertia around $z$ axis, $J_{zz}$ | 0.017 $kgm^2$ |
| Thrust coefficient, $C_T$ | $4.9782\times10^{-8}$ $N/(rpm)^2$ |
| Torque coefficient, $C_Q$ | $8\times10^{-10}$ $Nm/(rpm)^2$ |

Even though the OptiTrack measurements were used instead of IMU measurements to calculate the derivatives of position and orientation, these data also appeared to be noisy, because the differentiation also amplified the noise in OptiTrack measurements. In addition, it was observed that the motor velocities also had a significant noise, as well as a considerably higher multiplication factor of 64 when converting the measurements into *rpm*. Hence, the OptiTrack data and the motor velocity measurements were filtered using Kalman filters before modelling the dynamic system.

### 4.2.1 Measurement Filtering

#### 4.2.1.1 Position Measurement Filter

The position measurements from the OptiTrack motion capture system was filtered to obtain smoother measurements for $p$, $v$ and $a$. A discretized Kalman filter [31] with the states, position ($\boldsymbol{p}$), velocity ($\boldsymbol{v}$), acceleration ($\boldsymbol{a}$) and jerk ($\boldsymbol{j}$) was used for filtering. The dynamic model is presented in (4.4). The first derivative of the jerk ($\dot{\boldsymbol{j}}$) was modelled as a zero-mean Gaussian noise, $\boldsymbol{\eta}_j \sim \mathcal{N}(0, 0.001\boldsymbol{I}_3)$. The position measurements were also assumed to be corrupted by a zero-mean Gaussian noise, $\boldsymbol{\eta}_p \sim \mathcal{N}(0, 0.0002\boldsymbol{I}_3)$. Here, $\boldsymbol{I}_3$ is the $3 \times 3$ identity matrix and covariances 0.001 and 0.0002 were determined by trial and error when tuning the filter.

$$\boldsymbol{x} = [\boldsymbol{p}, \boldsymbol{v}, \boldsymbol{a}, \boldsymbol{j}]^T \tag{4.4a}$$

$$\dot{\boldsymbol{x}} = [\boldsymbol{v}, \boldsymbol{a}, \boldsymbol{j}, \boldsymbol{\eta}_j]^T \tag{4.4b}$$

With the measurements,

$$\boldsymbol{y}_p = \boldsymbol{p} + \boldsymbol{\eta}_p \tag{4.5}$$

Once the dynamic model is determined, the Kalman filter algorithm was followed. In order to proceed with Kalman filter, the process noise covariance matrix $\boldsymbol{Q}$ and measurement noise covariance matrix $\boldsymbol{R}$ were defined as,

$$\boldsymbol{Q} = 100 \begin{bmatrix} \mathbf{0}_{9\times9} & \mathbf{0}_{9\times3} \\ \mathbf{0}_{3\times9} & 0.1\boldsymbol{I}_3 \end{bmatrix} \tag{4.6a}$$

$$\boldsymbol{R} = \begin{bmatrix} 0.0002^2 & 0 & 0 \\ 0 & 0.0002^2 & 0 \\ 0 & 0 & 0.0002^2 \end{bmatrix} \tag{4.6b}$$

The state-space model of this process is defined by (4.7). This is the prediction stage in the Kalman filter, where the next state estimation, $\hat{\boldsymbol{x}}$, and covariance, $\boldsymbol{P}$, are propagated.

$$\hat{\boldsymbol{x}}_k = \boldsymbol{\Phi}\hat{\boldsymbol{x}}_{k-1} \tag{4.7a}$$

$$\boldsymbol{P}_k^- = \boldsymbol{\Phi}\boldsymbol{P}_{k-1}\boldsymbol{\Phi}^T + \boldsymbol{Q}, \tag{4.7b}$$

The discrete-time state transition matrix $\boldsymbol{\Phi}$ is defined as,

$$\Phi = \begin{bmatrix} \boldsymbol{I}_3 & dt\boldsymbol{I}_3 & \frac{dt^2}{2}\boldsymbol{I}_3 & \frac{dt^3}{6}\boldsymbol{I}_3 \\ \mathbf{0}_3 & \boldsymbol{I}_3 & dt\boldsymbol{I}_3 & \frac{dt^2}{2}\boldsymbol{I}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \boldsymbol{I}_3 & dt\boldsymbol{I}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \boldsymbol{I}_3 \end{bmatrix} \tag{4.8}$$

where, $dt$ represents the discrete sampling time.

Next, the observer correction in (4.9) is performed on measurements.

$$\hat{\boldsymbol{y}}_k = \boldsymbol{H}\hat{\boldsymbol{x}}_k \tag{4.9a}$$

$$\boldsymbol{K} = \boldsymbol{P}_k^- \boldsymbol{H}^T (\boldsymbol{H}\boldsymbol{P}_k^- \boldsymbol{H}^T + \boldsymbol{R})^{-1} \tag{4.9b}$$

$$\boldsymbol{x}_k = \boldsymbol{x}_k + \boldsymbol{K}(\boldsymbol{y}_k - \hat{\boldsymbol{y}}_k) \tag{4.9c}$$

$$\boldsymbol{P}_k^+ = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{H})\boldsymbol{P}_k^- \tag{4.9d}$$

Here, $\boldsymbol{H}$ is the output matrix and $K$ is the Kalman gain. The $\boldsymbol{H}$ matrix is defined as follows.

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{I}_3 & \boldsymbol{0}_{3\times 9} \end{bmatrix} \tag{4.10}$$

This process was followed on every OptiTrack position measurement. After iterating through the dataset, the filtered $\boldsymbol{p}$, $\boldsymbol{v}$ and $\boldsymbol{a}$ data were generated. The results after filtering is illustrated in Figures 4.2, 4.3 and 4.4. As the figures show, the noises incorporated with the velocity and acceleration measurements have been reduced, resulting a smoother output. These results were used for the learning process described in section 4.3.

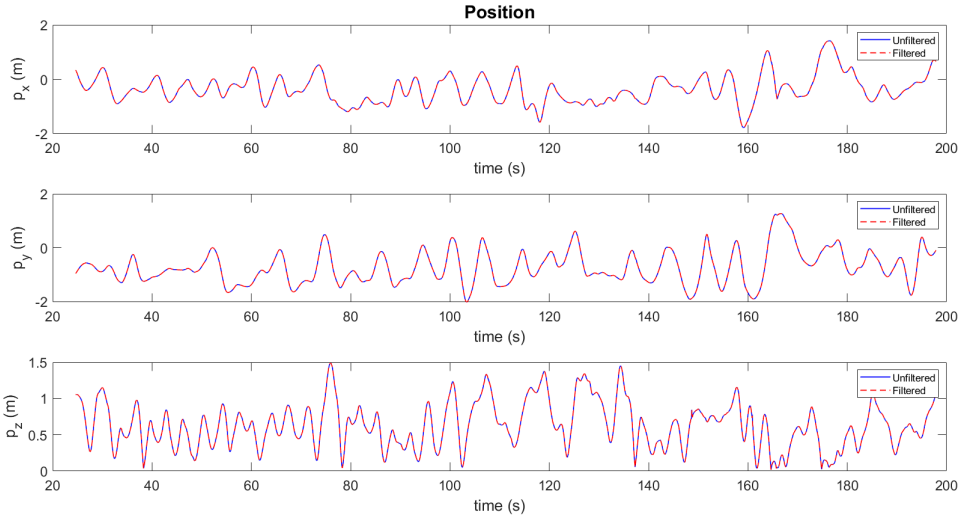



Figure 4.2: Filtered position, $\boldsymbol{p}$

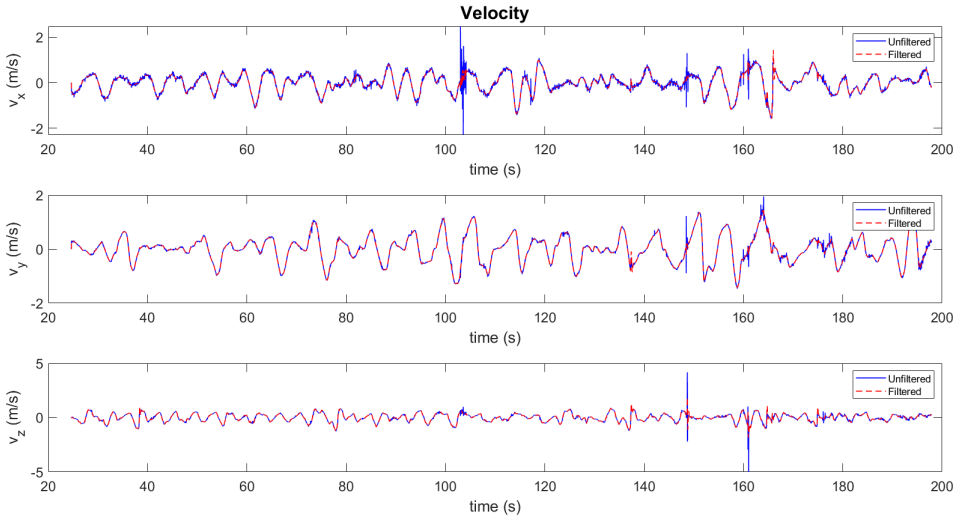



Figure 4.3: Filtered velocity, $\boldsymbol{v}$

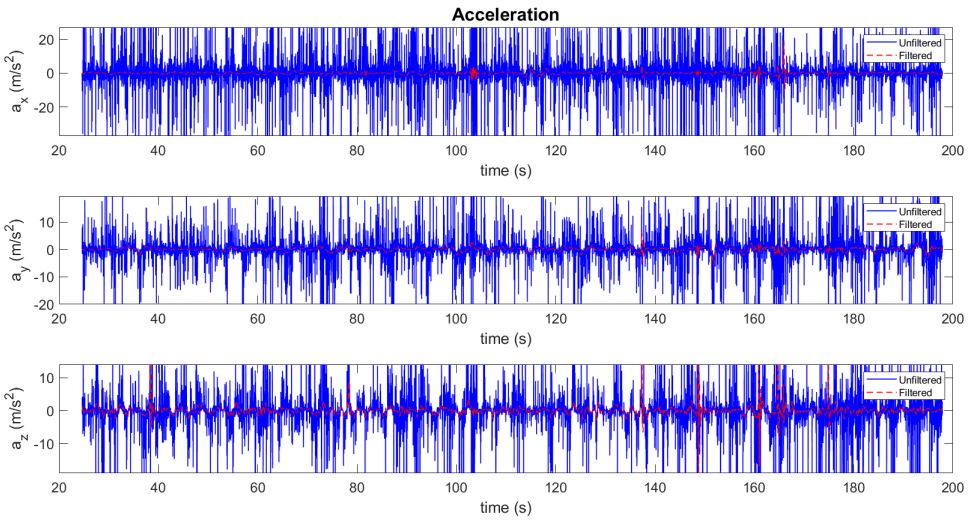



Figure 4.4: Filtered acceleration, $\boldsymbol{a}$

#### 4.2.1.2 Orientation Measurement Filter

The OptiTrack orientation quaternion measurement, $\boldsymbol{q}$ and its derivatives were also filtered using an error-state Kalman filter (ESKF) [32]. Unlike the position filter in section 4.2.1.1, the error in state vector and the error in output vector of the rotations do not have a linear behaviour. These rotations belong to the matrix Lie group $SO(3)$ [33]. If the error between the actual state ($\boldsymbol{R} \in SO(3)$) and the predicted state ($\hat{\boldsymbol{R}} \in SO(3)$) is expressed as $\boldsymbol{R} - \hat{\boldsymbol{R}}$, this result does not belong to $SO(3)$. This issue was addressed by defining an exponential map from a small perturbation of rotation error $\delta\theta \in \mathbb{R}^3$ in a direction $\mathbf{u}$ to the rotation in $SO(3)$ [32] as follows.

$$\exp(\delta\theta\mathbf{u}) = \boldsymbol{I}\cos(\delta\theta) + [\mathbf{u}]_\times \sin(\delta\theta) + \mathbf{u}\mathbf{u}^T(1 - \cos(\delta\theta)) \tag{4.11}$$

Hence,

$$\boldsymbol{R} = \exp(\delta\theta\mathbf{u}). \tag{4.12}$$

To inverse this exponential map, that is to convert from SO(3) to $\boldsymbol{R}$, a logarithmic map is defined as below.

$$\delta\theta\mathbf{u} = \log(\boldsymbol{R}) \tag{4.13}$$

$\mathbf{u}$ and $\delta\theta$ is calculated from,

$$\delta\theta = \cos^{-1}\left(\frac{\text{trace}(\boldsymbol{R}) - 1}{2}\right) \tag{4.14a}$$

$$\mathbf{u} = \frac{(\boldsymbol{R} - \boldsymbol{R}^T)^\vee}{2\sin(\delta\theta)} \tag{4.14b}$$

Here, $(.)^\vee$ is the inverse operation of $[.]_\times$.

The concept of the exponential map is used in the ESKF to overcome the non-linearity in the rotation error. The states for this filter was chosen as orientation $\boldsymbol{q}$, angular velocity $\boldsymbol{\omega}$, angular acceleration $\boldsymbol{\alpha}$ and the first derivative of the angular acceleration $\dot{\boldsymbol{\alpha}}$.

$$\boldsymbol{x} = \left[\boldsymbol{q}, \boldsymbol{\omega}, \boldsymbol{\alpha}, \dot{\boldsymbol{\alpha}}\right]^T \tag{4.15}$$

The second derivative of $\boldsymbol{\alpha}$ was assumed to have a Gaussian noise $\boldsymbol{\eta}_{\ddot{\alpha}} \sim \mathcal{N}(0, 0.1\boldsymbol{I}_3)$. The measurements were taken as the rotation matrix obtained from the quaternion measurements with a Gaussian noise $\boldsymbol{\eta}_q \sim \mathcal{N}(0, 0.001\boldsymbol{I}_3)$. Here, $\boldsymbol{I}_3$ is the $3\times3$ identity matrix.

The process noise covariance matrix $\boldsymbol{Q}$ and measurement noise covariance matrix $\boldsymbol{R}_\nu$ for this filter were defined as,

$$\boldsymbol{Q} = 500 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and} \tag{4.16a}$$

$$\boldsymbol{R}_\nu = \begin{bmatrix} 0.001^2 & 0 & 0 \\ 0 & 0.001^2 & 0 \\ 0 & 0 & 0.001^2 \end{bmatrix}. \tag{4.16b}$$

The covariance of $\ddot{\boldsymbol{\alpha}}$ and $\boldsymbol{\eta}$, and the covariance matrices $\boldsymbol{Q}$ and $\boldsymbol{R}_\nu$ were determined by trial and error method when tuning the filter.

Since the measurements were propagated in terms of the rotation matrix $\boldsymbol{R}$, $\boldsymbol{q} = [q_w, \mathbf{q}_v]^T$ is converted into $\boldsymbol{R}$ as,

$$\boldsymbol{R} = (q_w^2 - \mathbf{q}_v^T \mathbf{q}_v)\boldsymbol{I} + 2\mathbf{q}_v \mathbf{q}_v^T + 2q_w[\mathbf{q}_v]_\times, \tag{4.17}$$

where $\boldsymbol{I}$ is the $3 \times 3$ identity matrix.

The state matrix in continuous domain $\boldsymbol{F}$ is defined as,

$$\boldsymbol{F} = \begin{bmatrix} \mathbf{0}_{3\times3} & \boldsymbol{R}^T & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \boldsymbol{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \boldsymbol{I}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix}. \tag{4.18}$$

As filter was executed in the discrete domain, this $\boldsymbol{F}$ matrix is converted into the discrete domain state matrix $\boldsymbol{\Phi}$ as follows.

$$\boldsymbol{\Phi} = \boldsymbol{I}_{12\times12} + \boldsymbol{F}dt + \boldsymbol{F}^2dt^2, \tag{4.19}$$

where $dt$ is the sampling time.

The process noise covariance matrix $\boldsymbol{Q}$ was also converted into the discrete domain as,

$$\boldsymbol{Q}_d = \boldsymbol{G}\boldsymbol{Q}\boldsymbol{G}^T dt^2. \tag{4.20}$$

Where,

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{I}_{3\times3} \end{bmatrix}^T. \tag{4.21}$$

Then the covariance $\boldsymbol{P}$ was propagated from the above matrices.

$$\boldsymbol{P}_k^- = \boldsymbol{\Phi}\boldsymbol{P}_{k-1}\boldsymbol{\Phi}^T + \boldsymbol{Q}_d. \tag{4.22}$$

When calculating the next state estimations, the rotations are integrated from the principles of the Lie group $SO(3)$. Since $\boldsymbol{\omega}$ and its higher derivatives still follow the rules in vector space, their derivatives were generated from the vector space integration laws.

$$\boldsymbol{q}_k = \exp(\boldsymbol{\omega}_{k-1}dt) \otimes \boldsymbol{q}_{k-1} \tag{4.23a}$$

$$\boldsymbol{\omega}_k = \boldsymbol{\omega}_{k-1} + \boldsymbol{\alpha}_{k-1}dt \tag{4.23b}$$

$$\boldsymbol{\alpha}_k = \boldsymbol{\alpha}_{k-1} + \dot{\boldsymbol{\alpha}}_{k-1}dt \tag{4.23c}$$

$$\dot{\boldsymbol{\alpha}}_k = \dot{\boldsymbol{\alpha}}_{k-1} \tag{4.23d}$$

where $\otimes$ indicates the quaternion multiplication and the exponential map exp(.) is

defined in (4.11).

In the observer correction step, an error term was defined as follows to correct the estimated states later in (4.27).

$$\boldsymbol{e} = \log(\boldsymbol{R}^T \boldsymbol{y}) \tag{4.24}$$

where $\boldsymbol{y}$ is the orientation measurement $\boldsymbol{q}$ converted into rotation matrix $\boldsymbol{R}$ (refer to (4.17)). The logarithmic map log(.) is defined in (4.13).

Kalman gain for the correction is calculated by

$$\boldsymbol{K} = \boldsymbol{P}_k^- \boldsymbol{H}^T (\boldsymbol{H} \boldsymbol{P}_k^- \boldsymbol{H}^T + \boldsymbol{R}_\nu)^{-1} \tag{4.25}$$

with the output matrix,

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{I}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} \end{bmatrix}. \tag{4.26}$$

The corrected state estimations and the covariance are,

$$\boldsymbol{q}_k = \boldsymbol{q}_k \otimes \exp\left(\boldsymbol{K}(1:3,:)\boldsymbol{e}\right) \tag{4.27a}$$

$$\boldsymbol{\omega}_k = \boldsymbol{\omega}_k + \boldsymbol{K}(4:6,:)\boldsymbol{e} \tag{4.27b}$$

$$\boldsymbol{\alpha}_k = \boldsymbol{\alpha}_k + \boldsymbol{K}(7:9,:)\boldsymbol{e} \tag{4.27c}$$

$$\dot{\boldsymbol{\alpha}}_k = \dot{\boldsymbol{\alpha}}_k + \boldsymbol{K}(10:12,:)\boldsymbol{e} \tag{4.27d}$$

$$\boldsymbol{P}_k^+ = \boldsymbol{P}_k^- - \boldsymbol{K}\boldsymbol{H}\boldsymbol{P}_k^- \tag{4.27e}$$

Following the above Kalman filter algorithm, the filtered measurements for $\boldsymbol{q}$, $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$ were generated. This result is presented in Figures 4.5, 4.7 and 4.6. These figures indicate that the noise is reduced considerably, but there are some regions with higher noisy data after the $5000^{th}$ data point. Since the dataset for learning was selected from $1500^{th}$ to $4500^{th}$ data point, these noisy data could be avoided.

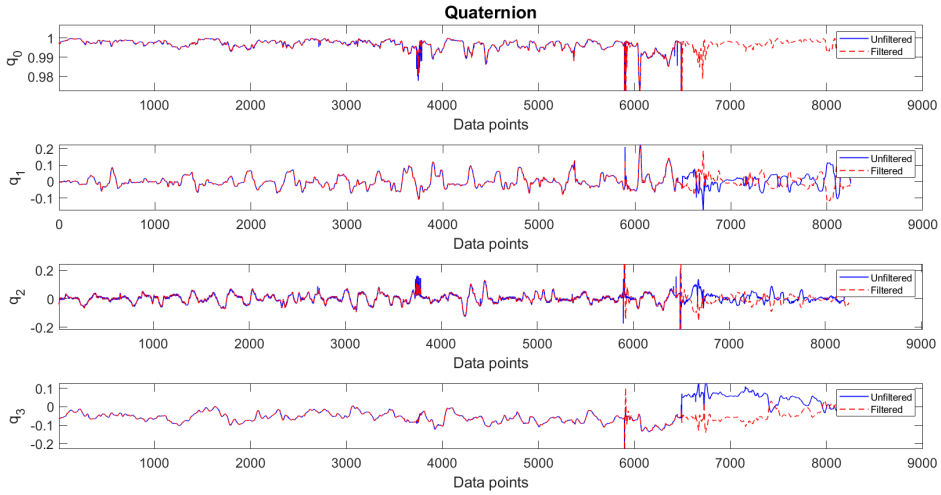



Figure 4.5: Filtered quaternion, $\boldsymbol{q}$

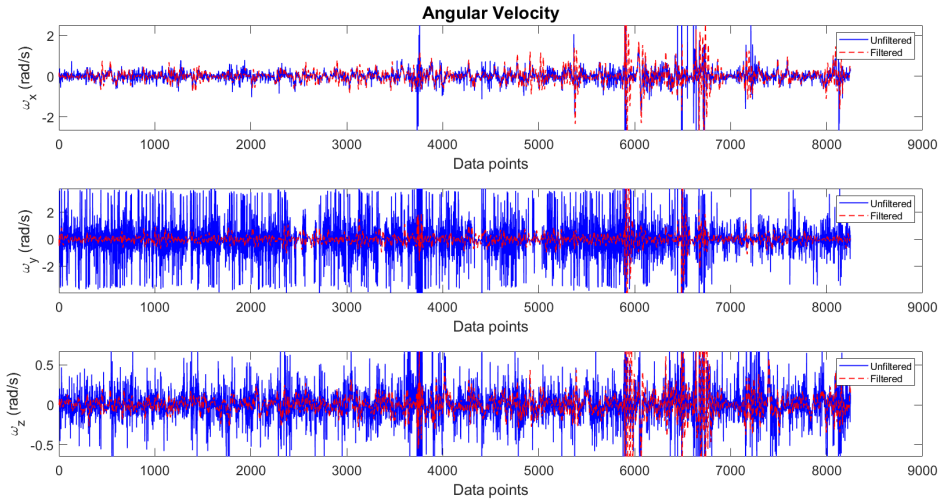



Figure 4.6: Filtered angular velocity, $\boldsymbol{\omega}$

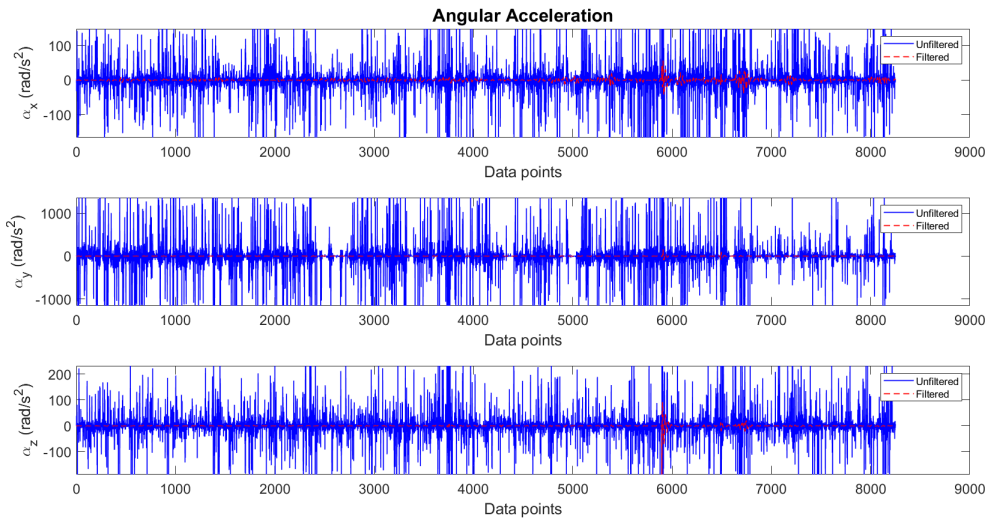



Figure 4.7: Filtered angular acceleration, $\boldsymbol{\alpha}$

#### 4.2.1.3 Motor Velocity Measurement Filter

It was observed that the motor velocity measurements were also corrupted by a noise. This resulted difficulties when training the unknown torque ${}^b\boldsymbol{\tau}_a$, because this noise is significant compared to ${}^b\boldsymbol{\tau}_a$. Hence, in addition to the OptiTrack measurements, the motor velocities were also filtered.

A discretized Kalman filter was used to filter the velocity measurements of each motor. The states were chosen as,

$$\boldsymbol{x} = [v_m, \dot{v}_m, \ddot{v}_m]^T \tag{4.28}$$

where, $v_m$, $\dot{v}_m$, $\ddot{v}_m$ are the motor velocity and first and second derivatives of the motor velocity, respectively. The third derivative of the motor velocity was modelled as a zero-mean Gaussian noise, $\eta_m \sim \mathcal{N}(0, 0.01)$. The dynamic model is shown in (4.29).

$$\dot{\boldsymbol{x}} = [\dot{v}_m, \ddot{v}_m, \eta_{\dot{v}m}]^T \tag{4.29}$$

Measurement of this model was chosen as $v_m$, with a zero-mean Gaussian noise, $\eta_m \sim \mathcal{N}(0, 1)$.

$$y = v_m + \eta_m \tag{4.30}$$

The discretized Kalman filter algorithm explained in section 4.2.1.1 was followed here as well. The process noise covariance matrix $\boldsymbol{Q}$ and measurement noise covariance matrix $\boldsymbol{R}$ were defined as,

$$\boldsymbol{Q} = 10000 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0.1 \end{bmatrix} \text{ and} \tag{4.31a}$$

$$\boldsymbol{R} = \begin{bmatrix} 1 \end{bmatrix}. \tag{4.31b}$$

Here, the higher order terms are neglected since the sampling rate is high.

The current state estimation and covariance propagation was done according to the state-space model defined in (4.7) and the observer correction was calculated from (4.9). The state transition matrix $\boldsymbol{\Phi}$ and output matrix $\boldsymbol{H}$ used in this filter are defined as follows.

$$\Phi = \begin{bmatrix} 1 & dt & \frac{dt^2}{2} \\ 0 & 1 & dt \\ 0 & 0 & 1 \end{bmatrix} \tag{4.32}$$

$$H = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \tag{4.33}$$

This methodology was followed for filtering the measurements of all four motors, having a separate filter for each motor with the same filtering matrices. The filtered result is demonstrated in Figure 4.8.

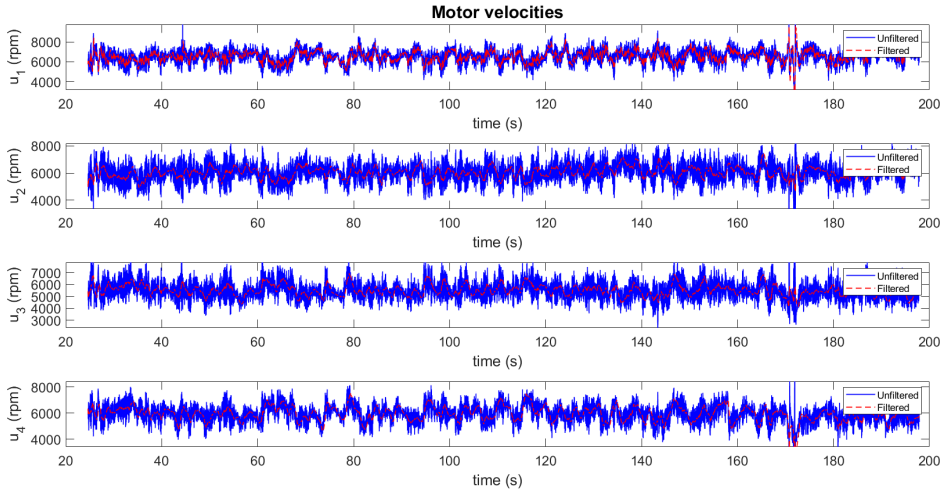



Figure 4.8: Filtered motor velocities

## 4.3 Learning Disturbance Forces

The main objective of this research is to learn the unmodelled force disturbances ${}^{b}\boldsymbol{f}_{a}$ in (4.1b). This section describes the methodology of learning the ${}^{b}\boldsymbol{f}_{a}$ using ANN. As described in Chapter 3, the dominant unmodelled force, i.e., the drag force has a linear relationship with the body-frame velocity. Therefore, the body frame force model was selected so that the body frame disturbance ${}^{b}\boldsymbol{f}_{a}$ will be learnt.

The training and testing sets were selected from the captured dataset in section 4.2, avoiding the sudden impacts as much as possible. The dataset from 1500 to 4500 data points were selected in this case, having the first 1500 points as the training set and the next 1500 points as the testing set. This way, the highly noisy data after 120$s$

(refer to Figure 4.3) could be eliminated. This selected dataset in the quadrotor's path is displayed in Figure 4.1.

The observed output ${}^{b}\boldsymbol{f}_a$ for the training set was calculated from this dataset by re-arranging (4.1b) as,

$$ {}^{b}\boldsymbol{f}_a = m{}^{b}\boldsymbol{a} - (m{}^{w}\boldsymbol{R}_b^T\boldsymbol{g} + {}^{b}\boldsymbol{T}). \tag{4.34} $$

The same equation was used to calculate the expected output of the testing set for comparison with the ANN prediction. The measurements ${}^{b}\boldsymbol{a}$ and ${}^{w}\boldsymbol{R}_b$ were calculated from the filtered data, while the thrust ${}^{b}\boldsymbol{T}$ was calculated from the motor velocities according to (4.3).

The body-frame velocity ${}^{b}\boldsymbol{v}$ was selected as the input for the NN in the initial approach described in section 3.2. That NN comprises of a single hidden layer with three hidden neurons and linear activation. This NN could reduce the model error from $0.563N$ to $0.5299N$, but this result was generated assuming that the drag force is the dominant force in ${}^{b}\boldsymbol{f}_a$. Although the results showed this assumption was correct, the model still had errors due to other complex aerodynamic effects and the system imperfections. Since the quadrotor's physical parameters were adopted from literature, it can be considered as a main reason to the system imperfections. In order to overcome these issues, the NN was designed more complex, by increasing the number of inputs, hidden layers, hidden layer neurons as well as the activation functions.

In addition to the model states, the model inputs also affect the system dynamics. Therefore, including the system input $\boldsymbol{u}$ as an input to the NN will result a better prediction than using only ${}^{b}\boldsymbol{v}$ as the input. Adding higher order linear terms (${}^{b}\boldsymbol{a}$) and orientation data ($\boldsymbol{q}$, ${}^{b}\boldsymbol{\omega}$, ${}^{b}\boldsymbol{\alpha}$) to the input might also capture the effect of these terms as well. By increasing the number of hidden layers the complexity of the model

can be increased. The activation function can be changed to introduce non-linearities to the model as well. Since several literature that used ReLU activation has proven successful results, [8, 18, 21] this function was evaluated in this work as well.

Considering all of these requirements, the prediction was generated for different ANN architectures and model inputs. The RMSE in the testing set of all these combinations was compared to select the best ANN architecture with the minimum RMSE. This comparison is presented in Table 4.2. Result for the ANN architecture used in [8] is presented as the first model in the table.

Table 4.2: Performance comparison of ANN architectures for learning ${}^{b}f_{a}$

| Model No. | Inputs | No. of hidden layers | No. of hidden layer neurons | Activation function | RMSE (N) |
|---|---|---|---|---|---|
| 0 | Nominal model | - | - | - | 0.5630 |
| 1 | ${}^{b}\boldsymbol{v}$, $p_z$, $\boldsymbol{q}$, $\boldsymbol{u}$ | 4 | 3 | *ReLU* | 0.5226 |
| 2 | ${}^{b}\boldsymbol{v}$ | 1 | 3 | *Linear* | 0.5264 |
| 3 | ${}^{b}\boldsymbol{v}$ | 1 | 3 | *ReLU* | 0.5210 |
| 4 | ${}^{b}\boldsymbol{v}$, $\boldsymbol{u}$ | 1 | 3 | *Linear* | 0.4190 |
| 5 | ${}^{b}\boldsymbol{v}$, $\boldsymbol{u}$ | 1 | 5 | *Linear* | 0.4193 |
| 6 | ${}^{b}\boldsymbol{v}$, $\boldsymbol{u}$ | 1 | 5 | *ReLU* | 0.4417 |
| 7 | ${}^{b}\boldsymbol{v}$, $\boldsymbol{u}$ | 2 | 2 | *Linear* | 0.4333 |
| 8 | ${}^{b}\boldsymbol{v}$, $\boldsymbol{u}$ | 2 | 3 | *Linear* | 0.4139 |
| 9 | ${}^{b}\boldsymbol{v}$, $\boldsymbol{u}$ | 2 | 4 | *Linear* | 0.4170 |
| 10 | ${}^{b}\boldsymbol{v}$, $\boldsymbol{u}$ | 2 | 5 | *Linear* | 0.4206 |
| 11 | ${}^{b}\boldsymbol{v}$, $\boldsymbol{u}$ | 2 | 5 | *ReLU* | 0.5175 |
| 12 | ${}^{b}\boldsymbol{v}$, $\boldsymbol{u}$ | 3 | 5 | *Linear* | 0.4158 |
| 13 | ${}^{b}\boldsymbol{v}$, $\boldsymbol{u}$ | 3 | 3 | *Linear* | 0.4178 |
| 14 | ${}^{b}\boldsymbol{v}$, ${}^{b}\boldsymbol{a}$, $\boldsymbol{u}$ | 2 | 5 | *Linear* | 0.2542 |
| 15 | ${}^{b}\boldsymbol{v}$, ${}^{b}\boldsymbol{a}$, $\boldsymbol{u}$ | 2 | 5 | *ReLU* | 0.3071 |
| 16 | ${}^{b}\boldsymbol{v}$, ${}^{b}\boldsymbol{a}$, $\boldsymbol{p}$, $\boldsymbol{u}$ | 2 | 5 | *ReLU* | 0.3009 |
| 17 | ${}^{b}\boldsymbol{v}$, ${}^{b}\boldsymbol{a}$, $\boldsymbol{p}$, $\boldsymbol{u}$ | 2 | 5 | *Linear* | 0.2389 |
| 18 | ${}^{b}\boldsymbol{v}$, ${}^{b}\boldsymbol{a}$, $\boldsymbol{q}$, $\boldsymbol{u}$ | 2 | 5 | *Linear* | 0.0775 |
| 19 | ${}^{b}\boldsymbol{v}$, $\boldsymbol{q}$, $\boldsymbol{u}$ | 2 | 5 | *Linear* | 0.4190 |
| 20 | ${}^{b}\boldsymbol{v}$, $\boldsymbol{q}$, $\boldsymbol{\omega}$, $\boldsymbol{u}$ | 2 | 5 | *Linear* | 0.4268 |

These results prove that introducing the motor velocities $\boldsymbol{u} = [u_1, u_2, u_3, u_4]$ to the NN inputs has significantly reduced the RMSE (refer to model numbers 2 and 4). Furthermore, the RMSE has been decreased by adding the higher order terms and the orientations (refer to model numbers 13, 14 and 18). In general, when introducing the non-linearities by $ReLU$ activation function, the RMSE has been slightly increased than using *linear* activation. Only the $3^{rd}$ configuration has the opposite behaviour. Therefore, it was concluded that adding non-linearities from $ReLU$ activation does not have a positive effect in this work. The reason for this behaviour is that the relationship between the dominant component of the unmodelled force ${}^b\boldsymbol{f}_a$ (drag force) and the input ${}^b\boldsymbol{v}$ is linear.

When increasing the number of layers and number of hidden neurons, first the RMSE was decreased. After two hidden layers and three hidden neurons, the RMSE started increase due to over-fitting. We compared this result when using ${}^b\boldsymbol{v}$ and $\boldsymbol{u}$ as inputs.

The least RMSE was obtained by the $18^{th}$ model, which used the acceleration and orientation quaternion in addition to ${}^b\boldsymbol{v}$ and $\boldsymbol{u}$. The RMSE in this model is 13.8% of the nominal model. This model is consisted of two hidden layers with five neurons in each hidden layer. Since the linear activation function is used here, the overall output of this ANN can be expressed by combining the weight and bias matrices in each layer as,

$$ {}^b\boldsymbol{f}_a = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \tag{4.35} $$

where $\boldsymbol{x} = [{}^b\boldsymbol{v}, {}^b\boldsymbol{a}, \boldsymbol{q}, \boldsymbol{u}]^T$ is the input vector to the ANN.

The weight matrix $\boldsymbol{W}$ and bias vector $\boldsymbol{b}$ are calculated by,

$$ \boldsymbol{W} = \boldsymbol{W}_2\boldsymbol{W}_1\boldsymbol{W}_0 \tag{4.36a} $$

where, $\boldsymbol{W}_i$ denotes the weight matrix of $i^{th}$ layer, including the input layer (layer 0).

$$\boldsymbol{b} = \boldsymbol{W}_2\boldsymbol{W}_1\boldsymbol{b}_0 + \boldsymbol{W}_2\boldsymbol{b}_1 + \boldsymbol{b}_2 \tag{4.36b}$$

where, $\boldsymbol{b}_i$ denotes the bias vector of $i^{th}$ layer, including the input layer (layer 0).

These $\boldsymbol{W}$ and $\boldsymbol{b}$ matrices were trained as follows.

$$\boldsymbol{W} = \begin{bmatrix} 0.0035 & -0.0016 & -0.0079 & 0.9190 & -0.1057 & -0.0015 & 0.0348 \\ -0.0215 & -0.0219 & -0.0145 & 0.1949 & 1.1993 & 0.1803 & -0.0058 \\ 0.0006 & -0.0023 & 0.0019 & -0.0025 & -0.0069 & 0.6978 & 0.0081 \\ -0.1317 & -0.8695 & -0.0242 & -0.0022 & -0.0048 & -0.0077 & 0.0011 \\ 1.0527 & -0.1820 & 0.0015 & -0.1105 & 0.0202 & -0.0433 & -0.0078 \\ -0.0066 & 0.0033 & -0.0059 & -0.1770 & -0.0853 & -0.1281 & -0.1578 \end{bmatrix} \tag{4.37a}$$

$$\boldsymbol{b} = \begin{bmatrix} 0.5036 & 0.2560 & 0.0042 \end{bmatrix}^T \tag{4.37b}$$

The result of this ANN was compared with the expected output calculated from (4.34). Figure 4.9 shows the predicted output from ANN plotted against the calculated expected output. The first 1500 data points represent the training set, while the next 1500 points belong to the testing set.

Although this ANN model showed the least RMSE among the compared models, when using this model in the simulation in Chapter 5, the system became highly unstable, with significantly larger trajectory errors. This issue was encountered when using the other ANN architectures with ${}^b\boldsymbol{a}$ as an input to the network. A stability analysis was carried out in section 5.3.1 to check the impact of having ${}^b\boldsymbol{a}$ in the NN model. The

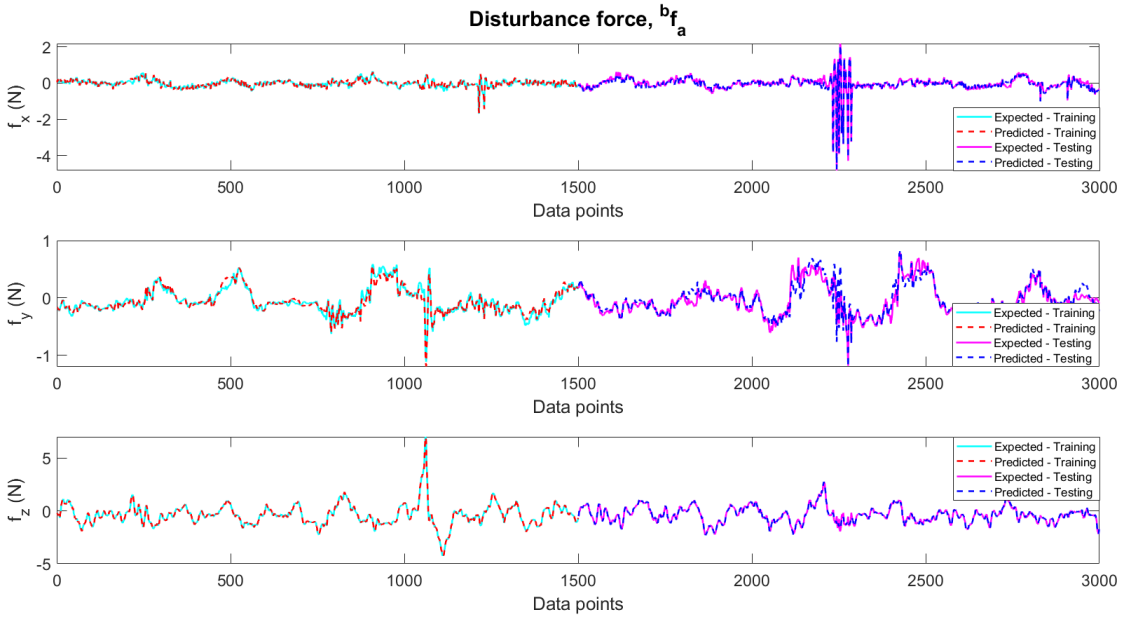



Figure 4.9: Disturbance force learnt from $18^{th}$ model

results show that the system becomes unstable when ${}^{b}\boldsymbol{a}$ is included as an input to the NN. Therefore, all network models with acceleration input was rejected.

Among the remaining architectures, the model with minimum RMSE (model 8) was chosen as the best model. This model has ${}^{b}\boldsymbol{v}$ and $\boldsymbol{u}$ as inputs, two hidden layers and three hidden layer neurons. Linear activation is used in this model as well. Since this network has the same number of hidden layers as the $18^{th}$ model, (4.36) can be used to calculate the weights and biases. The final weight and bias for this model are,

$$\boldsymbol{W} = \begin{bmatrix} -0.2053 & 0.0017 & 0.0156 & -0.0242 & 0.0473 & 0.0474 & -0.1034 \\ -0.0992 & -0.4139 & -0.0479 & -0.2134 & 0.0402 & -0.0952 & 0.1189 \\ -0.0387 & 0.0313 & 0.0203 & -0.2092 & -0.0329 & -0.0251 & -0.2713 \end{bmatrix} \tag{4.38a}$$

$$\boldsymbol{b} = \begin{bmatrix} 0.4768 & 0.2994 & -0.3817 \end{bmatrix}^{T} \tag{4.38b}$$

The learnt predictions for ${}^{b}\boldsymbol{f}_a$ using the $8^{th}$ model is shown in Figure 4.10.

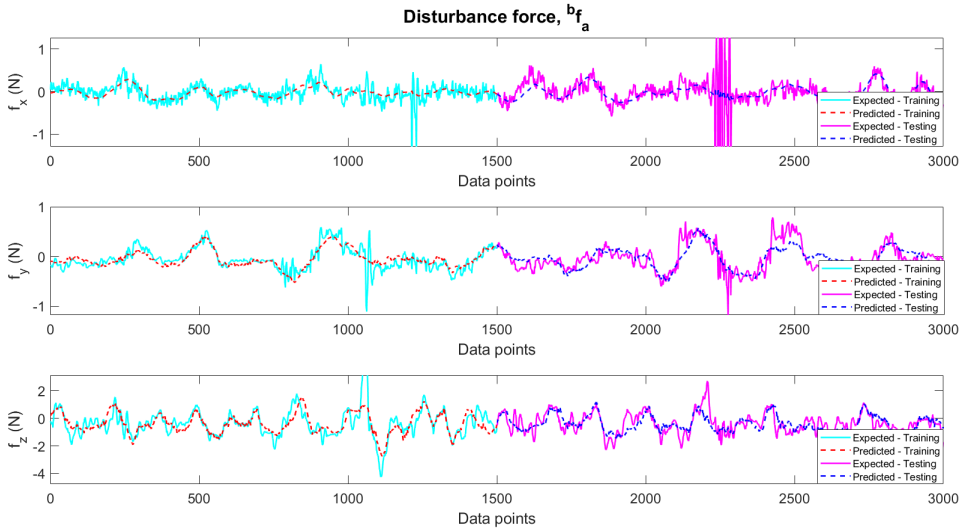



Figure 4.10: Disturbance force learnt from $8^{th}$ model

Although the prediction from the $8^{th}$ model is not accurate as the $18^{th}$ model, the $8^{th}$ model also tracks the desired ${}^{b}\boldsymbol{f}_a$ significantly. This model has neglected the nigher noisy data between $2000^{th}$ point to $2500^{th}$ point.

The overall RMS error in this model is 73.5% of the RMSE in the nominal model. In addition, the RMSE of the force error components in $x$, $y$ and $z$ directions were also compared in Figure 4.11. According to the figure, RMSEs are significantly reduced once the learnt disturbance was added, resulting a notable improvement in the dynamic model.

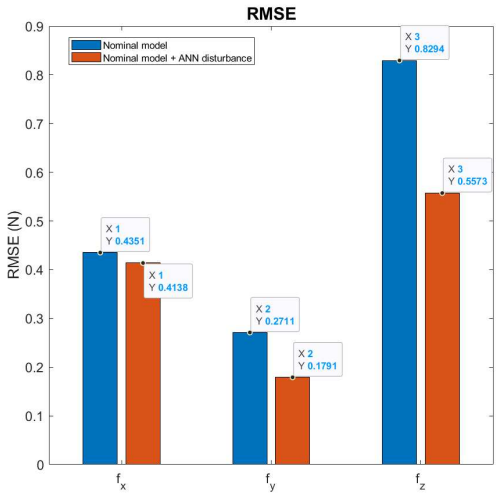



Figure 4.11: RMSE comparison of $8^{th}$ model

## 4.4 Learning Unmodelled Torque

The next objective of this work is to learn the unmodelled torque ${}^{b}\boldsymbol{\tau}_a$ in (4.1d). The procedure of designing an ANN and the results obtained for learning ${}^{b}\boldsymbol{\tau}_a$ is discussed in this section.

The same dataset used for learning the force disturbance in section 4.3 was selected in this approach as well. The first 1500 data points of this dataset were chosen as the training set, while the testing set was the next 1500 points. In order to calculate the observed output for training process and the expected output to compare with the ANN predictions while testing, (4.1d) was re-arranged as,

$$ {}^{b}\boldsymbol{\tau}_a = \boldsymbol{J}^{b}\boldsymbol{\alpha} - (J^{b}\boldsymbol{\omega} \times {}^{b}\boldsymbol{\omega} + {}^{b}\boldsymbol{\Gamma}). \tag{4.39} $$

This equation is expressed in body frame, therefore ${}^{b}\boldsymbol{\tau}_a$ will be learnt w.r.t. the body-frame of the quadcopter. The measurements ${}^{b}\boldsymbol{\alpha}$ and ${}^{b}\boldsymbol{\omega}$ were calculated from the filtered data in section 4.2.1.2, while the $\boldsymbol{J}$ is collected from the literature (refer to

Table 4.1). ${}^b\boldsymbol{\Gamma}$ was calculated from (4.3), using the filtered motor velocities in section 4.2.1.3.

Having the knowledge that the system inputs $\boldsymbol{u}$ improves the model (from the work in section 4.3), the initial inputs for the ANN were chosen as the orientation quaternion $\boldsymbol{q}$ and the motor velocities $\boldsymbol{u}$. The initial ANN was designed to have one hidden layer with three hidden neurons and linear activation function. After generating the observation set of the training data, these input-output pairs were fed into the ANN to learn the ${}^b\boldsymbol{\tau}_a$ model. Then, the predicted outputs were generated for the testing set by feeding the testing inputs and its output set was compared with the expected values calculated from (4.39). This resulted the RMSE to reduce from $0.0841Nm$ to $0.00083Nm$.

Although the above network could reduce the model error, that might not be the best ANN model for this task. Therefore, different ANN architectures were tested having different combinations of inputs, increasing the complexity by changing the number of hidden layers and neurons and adding non-linearities from ReLU activation function. Table 4.3 shows the comparison of RMSE of the tested ANNs.

According to this analysis, it is noted that introducing the ReLU activation function for non-linearities does not improve the performance. Therefore, linear activation function was considered more suitable for this process. Among all architectures tested, the $13^{th}$, $14^{th}$, and $15^{th}$ models show the least RMSE. Among these models, the $13^{th}$ model was chosen because it has lesser number of hidden neurons, which will be faster than the other models when learning. This model uses $\boldsymbol{x} = [\boldsymbol{q}, {}^b\boldsymbol{\omega}, {}^b\boldsymbol{\alpha}, \boldsymbol{u}]^T$ as inputs, has two hidden layers with three hidden neurons and linear activation. The output

Table 4.3: Performance comparison of ANN architectures for learning ${}^{b}\boldsymbol{\tau}_a$

| Model No. | Inputs | No. of hidden layers | No. of hidden layer neurons | Activation function | RMSE (Nm) |
|---|---|---|---|---|---|
| 0 | Nominal model | - | - | - | 0.0841 |
| 1 | $\boldsymbol{q}$, $\boldsymbol{u}$ | 1 | 3 | *Linear* | 0.0137 |
| 2 | $\boldsymbol{q}$, $\boldsymbol{u}$ | 1 | 5 | *Linear* | 0.0106 |
| 3 | $\boldsymbol{q}$, $\boldsymbol{u}$ | 1 | 5 | *ReLU* | 0.0117 |
| 4 | $\boldsymbol{q}$, $\boldsymbol{u}$ | 2 | 5 | *Linear* | 0.0105 |
| 5 | $\boldsymbol{q}$, $\boldsymbol{u}$ | 3 | 3 | *Linear* | 0.0105 |
| 6 | $\boldsymbol{q}$, $\boldsymbol{u}$ | 3 | 3 | *ReLU* | 0.0116 |
| 7 | $\boldsymbol{q}$, $\boldsymbol{u}$ | 3 | 5 | *Linear* | 0.0105 |
| 8 | $\boldsymbol{q}$, $\boldsymbol{u}$ | 3 | 5 | *ReLU* | 0.0105 |
| 9 | $\boldsymbol{q}$, ${}^{b}\boldsymbol{\omega}$, $\boldsymbol{u}$ | 1 | 3 | *Linear* | 0.0088 |
| 10 | $\boldsymbol{q}$, ${}^{b}\boldsymbol{\omega}$, $\boldsymbol{u}$ | 1 | 5 | *Linear* | 0.0124 |
| 11 | $\boldsymbol{q}$, ${}^{b}\boldsymbol{\omega}$, $\boldsymbol{u}$ | 2 | 3 | *Linear* | 0.0084 |
| 12 | $\boldsymbol{q}$, ${}^{b}\boldsymbol{\omega}$, ${}^{b}\boldsymbol{\alpha}$, $\boldsymbol{u}$ | 2 | 2 | *Linear* | 0.0236 |
| 13 | $\boldsymbol{q}$, ${}^{b}\boldsymbol{\omega}$, ${}^{b}\boldsymbol{\alpha}$, $\boldsymbol{u}$ | 2 | 3 | *Linear* | 0.00083 |
| 14 | $\boldsymbol{q}$, ${}^{b}\boldsymbol{\omega}$, ${}^{b}\boldsymbol{\alpha}$, $\boldsymbol{u}$ | 2 | 4 | *Linear* | 0.00083 |
| 15 | $\boldsymbol{q}$, ${}^{b}\boldsymbol{\omega}$, ${}^{b}\boldsymbol{\alpha}$, $\boldsymbol{u}$ | 2 | 5 | *Linear* | 0.00084 |
| 16 | $\boldsymbol{q}$, ${}^{b}\boldsymbol{\omega}$, ${}^{b}\boldsymbol{\alpha}$, $\boldsymbol{u}$ | 2 | 3 | *ReLU* | 0.0238 |
| 17 | $\boldsymbol{q}$, ${}^{b}\boldsymbol{\omega}$, ${}^{b}\boldsymbol{\alpha}$, $\boldsymbol{u}$ | 2 | 5 | *ReLU* | 0.0062 |

of this network ${}^{b}\boldsymbol{\tau}_a$ can be calculated from the learnt weight $\boldsymbol{W}$ and bias $\boldsymbol{b}$ as follows.

$$ {}^{b}\boldsymbol{\tau}_a = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \tag{4.40} $$

Since this model also has the ANN architecture similar to the ANN architecture in section 4.3, the $\boldsymbol{W}$ and $\boldsymbol{b}$ can be calculated from the weights and biases in each layer according to (4.36). The trained $\boldsymbol{W}$ and $\boldsymbol{b}$ of the ANN model for learning ${}^{b}\boldsymbol{\tau}_a$ are,

$$\boldsymbol{W} = \begin{bmatrix} -0.0010 & 0.0011 & 0.0002 & 0.0007 & 0.0007 & -0.0007 & 0.0011 \\ -0.0013 & -0.0003 & 0.0002 & 0.0004 & -0.0007 & -0.0002 & -0.0003 \\ -0.0000 & -0.0000 & -0.0000 & 0.0000 & -0.0000 & -0.0000 & 0.0000 \\ 0.0251 & 0.0010 & -0.0003 & -0.0008 & -0.4667 & 0.0003 & 0.8365 \\ 0.0005 & 0.0247 & -0.0007 & 0.6661 & 0.0007 & -0.4940 & -0.0008 \\ 0.0000 & 0.0000 & 0.9694 & 0.4324 & -0.2130 & 0.3211 & -0.3817 \end{bmatrix} \tag{4.41a}$$

and

$$\boldsymbol{b} = \begin{bmatrix} 0.0986 & -0.0077 & 0.1417 \end{bmatrix}^T . \tag{4.41b}$$

The ANN prediction for the testing set was calculated from (4.39) using these values for $\boldsymbol{W}$ and $\boldsymbol{b}$. The results are then compared with the expected output, as presented in Figure 4.12. This plot clearly indicates how the ANN prediction overlaps with the expected result.

The RMSE of the nominal model in (4.1d) and the RMSE of the model after adding the predicted disturbance from the ANN is compared in Figure 4.13. This result was generated for each dimension in ${}^b\boldsymbol{\tau}_a$. The overall RMSE is 0.98% of the nominal model.

## 4.5 Summary

The methodology and results of learning unmodelled quadrotor dynamics using ANN were included in this chapter. Several network architectures with different inputs were compared to select the best suitable model. When learning ${}^b\boldsymbol{f}_a$, the deep neural network with two hidden layers and five hidden neurons showed the least RMS error.

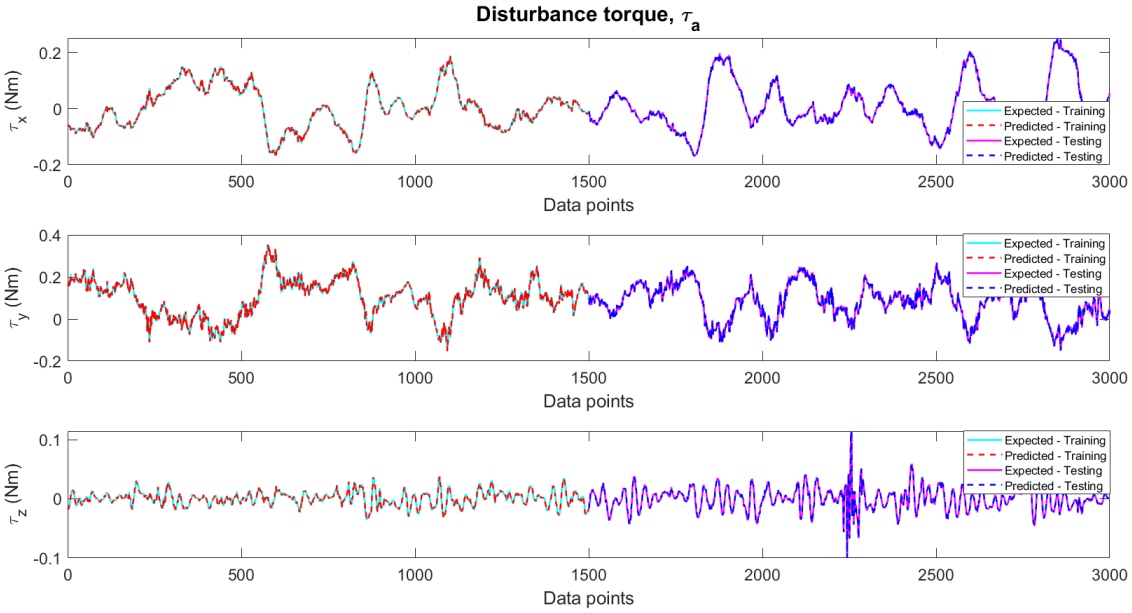



Figure 4.12: Disturbance learnt from $13^{th}$ model

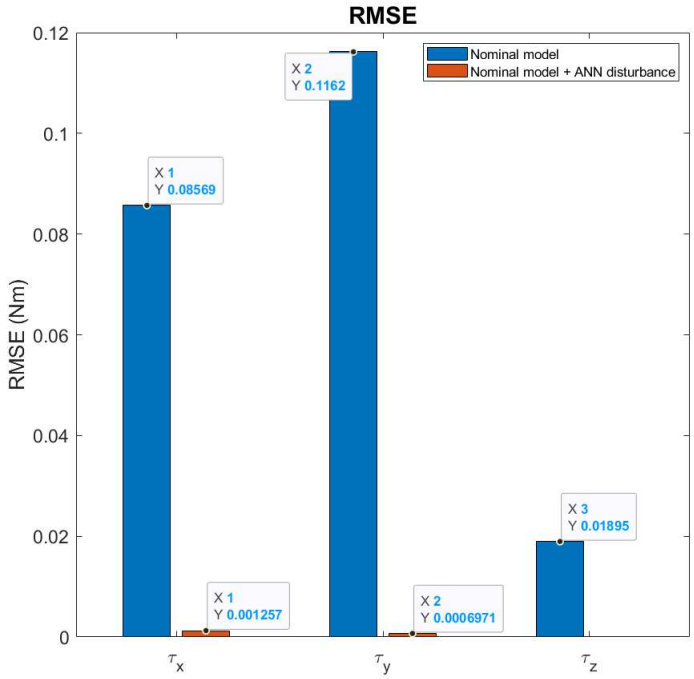



Figure 4.13: RMSE comparison of $13^{th}$ model

This network had the inputs ${}^b\boldsymbol{v}$, ${}^b\boldsymbol{a}$, $\boldsymbol{q}$ and $\boldsymbol{u}$. Although this network could reduce the model error to 13.7% of the nominal model error, higher instabilities were occurred when it was used in the simulation. The stability analysis proves that if the acceleration is taken as an input to the ANN, the system becomes unstable. Therefore, among the the ANN models which do not contain ${}^b\boldsymbol{a}$, the model with the least RMSE was chosen. This model is consisted of two hidden layers with linear activation, three hidden neurons and the inputs, ${}^b\boldsymbol{v}$ and $\boldsymbol{u}$. This model reduced the RMS error to 73.5% of the nominal model.

The unmodelled torque ${}^b\boldsymbol{\tau}_a$ was also learnt using an ANN. After the comparison of different NN architectures, the network with the least RMSE was chosen as the best model. The chosen model had two hidden layers with linear activation function and three hidden neurons. Inputs to the network were $\boldsymbol{q}$, ${}^b\boldsymbol{\omega}$, ${}^b\boldsymbol{\alpha}$ and $\boldsymbol{u}$. When this model was added to the nominal model, it could achieve an RMSE of $0.00083Nm$, which was 0.98% of the nominal model error.

In conclusion, the work carried out in this chapter could achieve successful results for learning ${}^b\boldsymbol{f}_a$ and ${}^b\boldsymbol{\tau}_a$ using ANN.

# Chapter 5

# Simulated Multi-rotor Aerial Vehicle (MAV) Control Using Learned Unknown Dynamics

This section includes the application of the learnt unknown dynamics in quadcopter controlling. A trajectory controller was designed with an added disturbance model. The improvement of the trajectory tracking when the learnt disturbance is included in the dynamic model was analyzed.

## 5.1 Overview of the Simulation

The simulated quadrotor control system in this thesis is designed to follow a pre-defined trajectory. Once the desired trajectory data is given, the controller generates the desired motor velocities to the four rotors. A disturbance is added to the system so that the ANN can learn the disturbance model to reduce the trajectory error.

The quadcopter control system is designed to have following states.

$$\boldsymbol{x} = [\boldsymbol{p}, \boldsymbol{v}, {}^{w}\boldsymbol{R}_{b}, \boldsymbol{\omega}]^{T} \tag{5.1}$$

The process of calculating the next states is as follows. The current states $\boldsymbol{x}$ and the desired states $\boldsymbol{x}_{des}$ are incorporated to calculate the desired Euler angles and thrust ($\boldsymbol{U}$) from a high-level LQR controller. This controller is described in section 5.2. The control input $\boldsymbol{u}$ to the quadcopter is then calculated using the desired Euler angles and thrust by a low-level proportional-derivative (PD) controller. The control input is referred to the velocities of the four rotors. Then, the desired thrust $\boldsymbol{T}$ along $z$-axis in the body frame and the desired torques $\boldsymbol{\Gamma}$ in the body frame are calculated from $\boldsymbol{u}$, according to (4.3). Since this force and torque are subjected to disturbances and model imperfection in practical scenarios, the disturbances should be modelled. In this simulation, the application of force disturbance is considered. The disturbance force in body frame w.r.t. the world frame expressed in body frame ${}^{b}\boldsymbol{f}_{a}$ is generated from $\boldsymbol{u}$ and $\boldsymbol{x}$, using an ANN. Using $\boldsymbol{T}$, $\boldsymbol{\Gamma}$ and ${}^{b}\boldsymbol{f}_{a}$, the state derivatives are determined from (5.2).

$$\begin{bmatrix} \dot{\boldsymbol{p}} \\ \dot{\boldsymbol{v}} \\ {}^{w}\dot{\boldsymbol{R}}_{b} \\ \dot{\boldsymbol{\omega}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{g} + ({}^{w}\boldsymbol{R}_{b}{}^{b}\boldsymbol{T} + {}^{w}\boldsymbol{R}_{b}{}^{b}\boldsymbol{f}_{a})/m \\ {}^{w}\boldsymbol{R}_{b}[\boldsymbol{\omega}]_{\times} \\ \boldsymbol{J}^{-1}(\boldsymbol{J}\boldsymbol{\omega} \times \boldsymbol{\omega} + {}^{b}\boldsymbol{\Gamma}) \end{bmatrix} \tag{5.2}$$

These derivatives and current states are incorporated to estimate the states in the next time step. The control inputs to the next iteration are computed using the next states and the desired states. This process is iteratively followed throughout the

trajectory. Note that the state $\boldsymbol{x}_0$ is initialized according to the trajectory data. The overview of this simulation is illustrated in Figure 5.1.

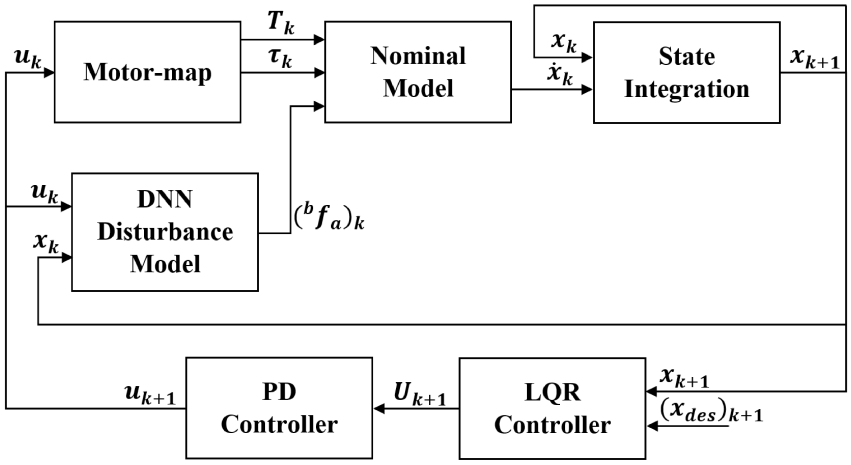



Figure 5.1: Overview of the simulation process

## 5.2 Simulated Trajectory and Model-based Controller

A simulated quadrotor trajectory controller was generated in MATLAB programming platform to demonstrate the improvement of the trajectory tracking with the learnt disturbance. An arbitrary trajectory within the learnt 3D space was generated for this simulation. It was defined by a sequence of waypoints and tangent vectors at those points. The following way points and tangent vectors were defined.

$$p = \begin{bmatrix} 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 1 & 2 & 0 & -1 & 0 \\ 0 & 0.8 & 1 & 0.5 & 1 & 0.8 \end{bmatrix} \tag{5.3a}$$

$$v = 0.5 \begin{bmatrix} 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.3b}$$

Each path segment between two consecutive waypoints were generated as a Bezier curve. To generate those Bezier curves, two intermediate control points ($p_1$ and $p_2$) were generated using the end point coordinates and tangents as follows.

$$p1 = p_{start} + v_{start}; \tag{5.4a}$$

$$p2 = p_{end} - v_{end}; \tag{5.4b}$$

where, $p_{start}$ and $p_{end}$ are the start and end waypoints in each segment and $v_{start}$ and $v_{end}$ are the tangent at that point. Using the four control points, a cubic Bezier curve was formed to obtain 1000 trajectory points in between.

$$p(t) = (1-t)^3 p_{start} + (1-t)^2 t p_1 + (1-t) t^2 p_2 + t^3 p_{end} \tag{5.5}$$

Here, $t$ refers to the $t^{th}$ sample. These spline positions were combined to generate the trajectory position, velocity and yaw direction. This path is indicated in Figures 5.2(a) and 5.3(a).

The quadcopter was controlled in this simulation using a high-level LQR controller and a low-level PD controller. The low-level controller represents the quadcopter's on-board controller. It generates the control inputs to the motors for desired Euler angles. The desired Euler angles are calculated by the high-level controller, using the

dynamic model. We selected the LQR controller as the high-level controller. In the LQR controller, an optimal control gain $\boldsymbol{K}$ is calculated such that a cost function is minimized.

The states of the LQR controller was defined as the position $\boldsymbol{p}$ and the velocity $\boldsymbol{v}$ and the acceleration $\boldsymbol{a}$ was chosen as the input. The system equations are given in (5.6).

$$\boldsymbol{x} = [\boldsymbol{p}, \boldsymbol{v}]^T \tag{5.6a}$$

$$\dot{\boldsymbol{x}} = [\boldsymbol{v}, \boldsymbol{\eta}_v]^T \tag{5.6b}$$

The state-space model of the LQR controller in the continuous domain is defined as,

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \tag{5.7}$$

where,

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{0}_{3\times3} & \boldsymbol{I}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} \end{bmatrix} \tag{5.8a}$$

and

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{0}_{3\times3} & \boldsymbol{I}_{3\times3} \end{bmatrix}^T \tag{5.8b}$$

For the LQR controller, the cost function in (5.9) was defined. We used the MATLAB in-built $lqr()$ function to calculate the optimal gain $\boldsymbol{K}$.

$$\boldsymbol{J} = \int_0^\infty [(\boldsymbol{x}_{des} - x)^T \boldsymbol{Q}(\boldsymbol{x}_{des} - \boldsymbol{x}) + \boldsymbol{u}^T \boldsymbol{R}\boldsymbol{u}]dt \tag{5.9}$$

In this simulation, the state error weight matrix $\boldsymbol{Q}$ and the control expenditure weight matrix $\boldsymbol{R}$ were defined as follows.

$$\boldsymbol{Q} = \begin{bmatrix} 100\boldsymbol{I}_{3\times3} & 10\boldsymbol{I}_{3\times3} \end{bmatrix} \tag{5.10a}$$

$$\boldsymbol{R} = 0.2\boldsymbol{I}_{3\times3} \tag{5.10b}$$

The optimal $\boldsymbol{K}$ was computed as,

$$\boldsymbol{K} = \begin{bmatrix} 22.3607 & 0.0000 & 0.0000 & 9.7325 & 0.0000 & -0.0000 \\ 0.0000 & 22.3607 & 0.0000 & 0.0000 & 9.7325 & 0.0000 \\ -0.0000 & 0.0000 & 22.3607 & -0.0000 & 0.0000 & 9.7325 \end{bmatrix}. \tag{5.11}$$

The desired acceleration $\boldsymbol{a}_{des}$ at each point in the trajectory was calculated as follows, using the $\boldsymbol{K}$ and trajectory acceleration $\boldsymbol{a}$.

$$\boldsymbol{a}_{des} = \boldsymbol{a} + \boldsymbol{K}(\boldsymbol{x}_{des} - \boldsymbol{x}) \tag{5.12}$$

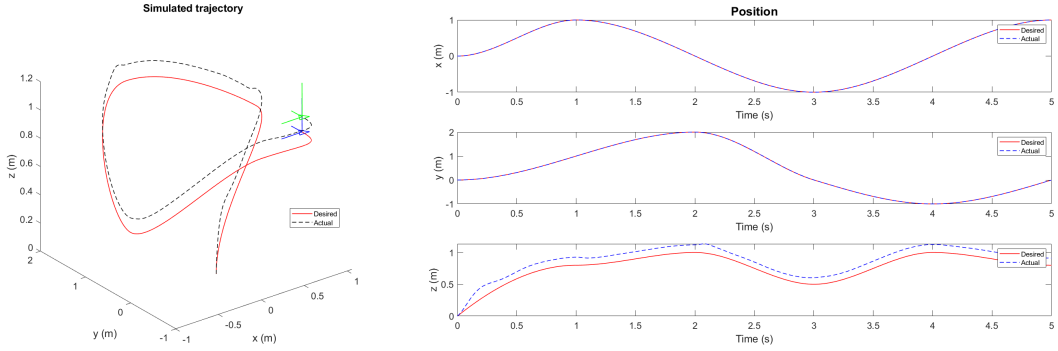
## 5.3 Performance of Quadrotor Trajectory Tracking with the Disturbance

As mentioned in section 5.2, the quadcopter was simulated to follow the defined trajectory with and without adding the disturbance to the dynamic model in the controller. In this simulation, the force disturbance was added. According to section 4.3, the ANN with the least RMS error was first selected as the best ANN model for this task. The system errors increased when using this model, resulting higher

instabilities. It was because the predicted ANN disturbance force using ${}^b\boldsymbol{a}$ was used to generate ${}^b\boldsymbol{a}$ in the next state. Therefore, the acceleration error was propagated throughout the simulation.

In order to overcome this problem, the ANN model recommended in section 4.3 was used to model the disturbances. This model had two hidden layers with three hidden neurons and linear activation. The inputs to the network were ${}^b\boldsymbol{v}$ and $\boldsymbol{u}$.

The dynamic model for the first controller was designed without adding the disturbance. Figure 5.2 presents the trajectory tracking result for this controller. It can be seen that the actual trajectory has a considerable tracking error in $z$ direction. The RMSE between actual and desired $x$, $y$ and $z$ positions were $0.0009m$, $0.0014m$ and $0.1104m$, respectively.

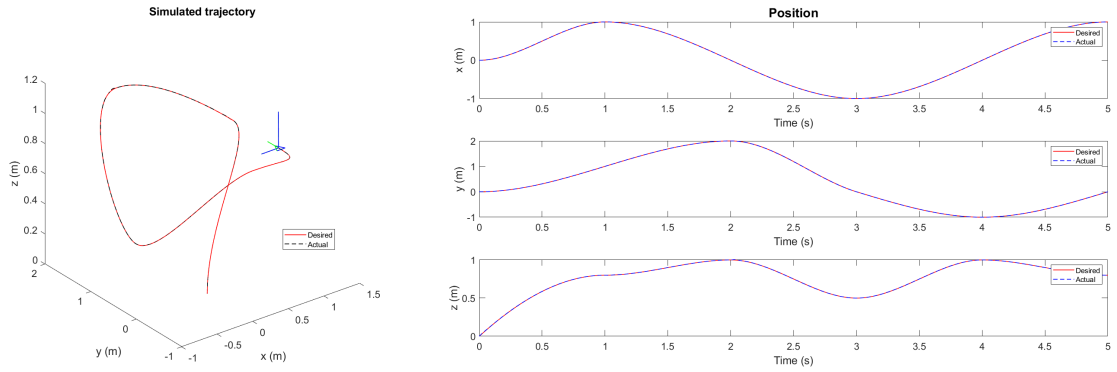



(a) Trajectory (b) Position

Figure 5.2: The trajectory followed by the controller without ANN model

In the second controller, the estimated disturbance force ${}^b\boldsymbol{f}_a$ was added to the dynamic model in the LQR controller. This controller could follow the trajectory with significantly lesser error than the previous controller. The results are shown in Figure 5.3. Note that the same PD controller was used in both controllers mentioned above and only the dynamics of the LQR controller were changed.

The actual Euler angles were also compared in Figure 5.4. This figure shows that the




(a) Trajectory (b) Position

Figure 5.3: The trajectory followed by the controller with ANN model

simulated quadcopter could follow the desired Euler angles with minimal error.

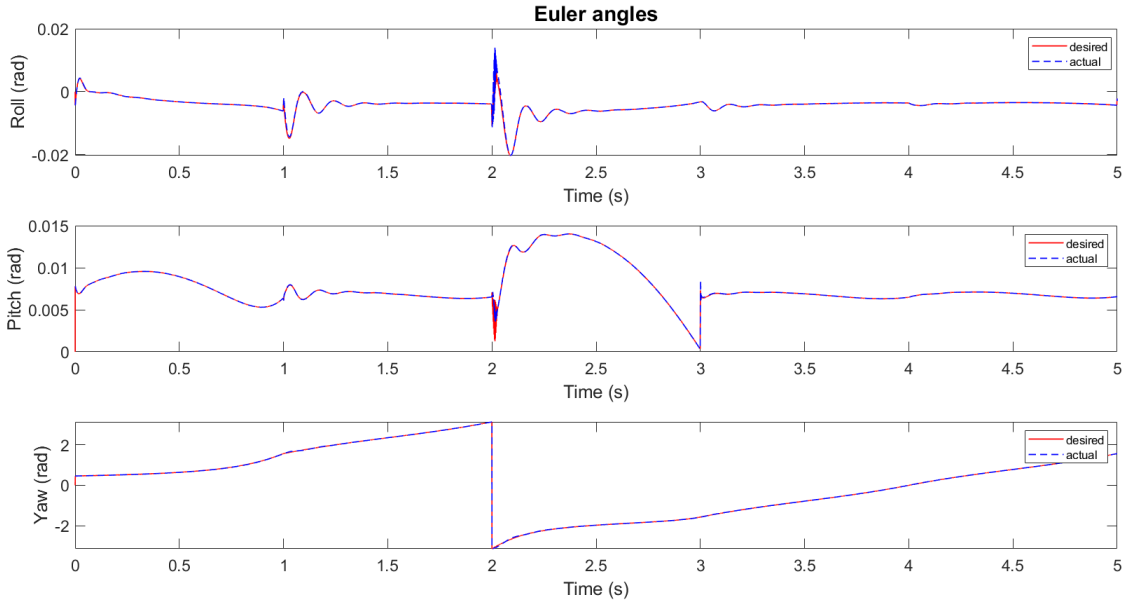



Figure 5.4: Comparison of actual and desired Euler angles

When the learnt unmodelled force ${}^b\boldsymbol{f}_a$ was included in the dynamics, the position RMSE was $0.0009m$, while the controller without ${}^b\boldsymbol{f}_a$ had an RMSE of $0.0729m$. The

plots and RMSE values conclude that the controller with the ANN model showed superior trajectory tracking performance in comparison to the controller without the ANN model.

### 5.3.1 Stability Analysis of the System when ${}^{b}\boldsymbol{a}$ is Included in the NN Inputs

Since all the ANN models with ${}^{b}\boldsymbol{a}$ input caused higher instabilities in the simulation, a stability analysis was carried out to check whether there is a stability issue when using ${}^{b}\boldsymbol{a}$ as an input to ANN.

Since the velocity is coupled with the acceleration, the state propagation of the velocity and acceleration is considered. The next state propagation of ${}^{w}\boldsymbol{v}$ and ${}^{w}\boldsymbol{a}$ can be expressed as,

$$\begin{bmatrix} {}^{w}\boldsymbol{v}_{k+1} \\ {}^{w}\boldsymbol{a}_{k+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I}_{(3\times3)} & \boldsymbol{I}_{(3\times3)}dt \\ \boldsymbol{0}_{(3\times3)} & \boldsymbol{0}_{(3\times3)} \end{bmatrix} \begin{bmatrix} {}^{w}\boldsymbol{v}_{k} \\ {}^{w}\boldsymbol{a}_{k} \end{bmatrix} + \underbrace{\frac{1}{m}\,{}^{w}\boldsymbol{R}_{b}\Big(\boldsymbol{W}\begin{bmatrix} {}^{b}\boldsymbol{v}_{k} \\ {}^{b}\boldsymbol{a}_{k} \\ \boldsymbol{q_k} \\ \boldsymbol{u}_{k} \end{bmatrix} + \boldsymbol{b}\Big)}_{\text{ANN prediction of } {}^{b}\boldsymbol{f}_{a}} + \begin{bmatrix} {}^{w}\boldsymbol{v}_{k,des} \\ {}^{w}\boldsymbol{a}_{k,des} \end{bmatrix} \tag{5.13}$$

where, ${}^{w}\boldsymbol{v}_{k,des}$ and ${}^{w}\boldsymbol{a}_{k,des}$ are desired velocity and acceleration, respectively. According to (5.13), ${}^{w}\boldsymbol{v}_{k+1}$ and ${}^{w}\boldsymbol{a}_{k+1}$ depends on the orientation and motor velocities as well, but here we are considering the effect of ${}^{w}\boldsymbol{v}_{k}$ and ${}^{w}\boldsymbol{a}_{k}$ only. Hence, (5.13) was re-arranged by separating the ${}^{w}\boldsymbol{v}_{k}$ and ${}^{w}\boldsymbol{a}_{k}$ terms as follows.

$$\begin{bmatrix} {}^{w}\boldsymbol{v}_{k+1} \\ {}^{w}\boldsymbol{a}_{k+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I}_{(3\times3)} & \boldsymbol{I}_{(3\times3)}dt \\ \frac{1}{m}{}^{w}\boldsymbol{R}_b W_v {}^{w}\boldsymbol{R}_b^T & \frac{1}{m}{}^{w}\boldsymbol{R}_b \boldsymbol{W}_a {}^{w}\boldsymbol{R}_b^T \end{bmatrix} \begin{bmatrix} {}^{w}\boldsymbol{v}_k \\ {}^{w}\boldsymbol{a}_k \end{bmatrix} + \frac{1}{m}{}^{w}\boldsymbol{R}_b \Big( \boldsymbol{W}_{q,u} \begin{bmatrix} \boldsymbol{q}_k \\ \boldsymbol{u}_k \end{bmatrix} + \boldsymbol{b} \Big) + \begin{bmatrix} {}^{w}\boldsymbol{v}_{k,des} \\ {}^{w}\boldsymbol{a}_{k,des} \end{bmatrix} \tag{5.14}$$

Here, $\boldsymbol{W}_v$ and $\boldsymbol{W}_a$ terms represent the portions of the weight matrix $\boldsymbol{W}$ corresponding to ${}^{w}\boldsymbol{v}_k$ and ${}^{w}\boldsymbol{a}_k$ respectively. Similarly, $\boldsymbol{W}_{q,u}$ term is the weight corresponding to $\boldsymbol{q}$ and $\boldsymbol{u}$. The coefficient matrix of $[{}^{w}\boldsymbol{v}_{k+1}, {}^{w}\boldsymbol{a}_{k+1}]^T$ is what we are interested in this stability analysis.

The quadcopter was unstable even at the hovering conditions. When hovering, ${}^{w}\boldsymbol{R}_b \approx \boldsymbol{I}$. Therefore, it is possible to simplify the coefficient matrix of $[{}^{w}\boldsymbol{v}_{k+1}, {}^{w}\boldsymbol{a}_{k+1}]^T$ by assuming ${}^{w}\boldsymbol{R}_b = \boldsymbol{I}$ as follows. Note that if it is possible to prove the system is unstable even when hovering, then the system is obviously unstable when moving as well.

$$\boldsymbol{M}_{v,a} = \begin{bmatrix} \boldsymbol{I}_{(3\times3)} & \boldsymbol{I}_{(3\times3)}dt \\ \frac{1}{m}\boldsymbol{W}_v & \frac{1}{m}\boldsymbol{W}_a \end{bmatrix} \tag{5.15}$$

The eigenvalues of this matrix were calculated to check whether the system is stable. Four of the six eigenvalues were greater than one, as shown below.

$$eig(\boldsymbol{M}_{v,a}) = \begin{bmatrix} 1.5810 + 0.1377i \\ 1.5810 - 0.1377i \\ 0.9984 + 0.0000i \\ 1.0017 + 0.0000i \\ 1.0000 + 0.0000i \\ 1.0001 + 0.0000i \end{bmatrix} \tag{5.16}$$

In addition, at least two of the eigenvalues were greater than one when checking the other models with ${}^{b}\boldsymbol{a}$ as an input as well. Therefore, we concluded that the system was unstable when acceleration was used as an input to the NN model.

## 5.4 Summary

Chapter 5 has evaluated the performance of a simulated quadrotor when the learnt unmodelled force was applied to the dynamics. A quadrotor trajectory tracking controller was simulated in this work. An arbitrary trajectory generated using Bezier curves was used as the desired trajectory. The quadrotor high-level and low-level controllers were an LQR controller and a PD controller, respectively. A disturbance was generated using the same ANN model chosen in Chapter 4. The trajectory tracking performance of the quadrotor was compared with and without adding ${}^{b}\boldsymbol{f}_a$ to the dynamics of the LQR controller. According to the results, the position error is reduced when using the controller with learnt disturbance force ${}^{b}\boldsymbol{f}_a$.

# Chapter 6

# Conclusions and Future Directives

This thesis is focused on learning unmodelled dynamics of a quadrotor UAV. Since the quadcopters are subjected to complex aerodynamic effects during the flight and model imperfections are present in the physical system, deriving complete dynamics are challenging. In order to improve the flight performance, these unmodelled dynamics are included in the dynamic model, so that the controller has a lesser error to minimize. When estimating these unmodelled dynamics, ML techniques have become a novel successful approach. The work in this thesis is based on learning the unmodelled dynamics using GPR and ANN. The following research objectives are identified for this research.

1. Evaluate GPR and ANN for estimating unmodelled dynamics of a quadcopter.
2. Model the unknown force and torque dynamics using ANN.
3. Demonstrate the improvement of trajectory tracking performance when the unknown dynamic model is incorporated in the controller.

## 6.1 Research Summery of Objective I

The first objective of this research was to evaluate and compare the application of GPR and ANN to learn the unmodelled dynamics of a quadcopter. Learning the unmodelled force error was considered in this comparison. A GPR model and an ANN model was learnt separately using the same dataset. The results showed that the GPR model had an RMS error of $0.6128N$ while the ANN model showed only $0.5273N$ error. This RMS error was calculated as the error in the force once the learnt dynamics are added. Although the literature says that GPR outperforms ANN, this comparison showed that ANN is more suitable than GPR for the work presented in this thesis. In addition, ANN has more flexibility than GPR. ANN can adapt to complex models by changing the number of hidden layers and number of hidden neurons, while the flexibility of GPR depends on the kernel function. Additionally, the performance of ANN was faster than GPR as well. Considering all the advantages, ANN was chosen for further analysis.

## 6.2 Research Summery of Objective II

The next objective was to further analyze the best approach, i.e., ANN for learning unmodelled force and torque dynamics. Assuming the non-linearity and complexity of the unknown model, the number of hidden layers, number of hidden layer neurons and the inputs to the ANN had to be changed. Different ANN architectures were compared by changing these parameters and the model with minimum RMS error was chosen as the best model.

For the force, the ANN model consisted of two hidden layers, five hidden layer neurons and ${}^b\boldsymbol{v}$, ${}^b\boldsymbol{a}$, $\boldsymbol{q}$ and $\boldsymbol{u}$ were used as inputs. When adding position and angular velocity

as inputs, the RMS error did not change considerably. Therefore, it is reasonable to say that the disturbance force does not have an effect on these variables. When the aforementioned model was included to the nominal model, the RMS error in the force was reduced to 13.8% of the nominal model.

However, when using that model in the simulation, the system showed higher instabilities. Since the learnt force is used to calculate the acceleration, the acceleration error is increased in each iteration. A stability analysis was carried out to check the stability of the controller if acceleration was used as an input. The stability analysis confirmed that instabilities occur when the acceleration was used as an input to the ANN. Therefore, an ANN model without acceleration input was considered. Among these models, the model with least RMS error was chosen. This model had two hidden layers, three hidden layer neurons and linear activation function. Inputs used in this model were ${}^{b}\boldsymbol{v}$ and $\boldsymbol{u}$. It could achieve an RMS error of 73.5% of the nominal model.

The ANN used for learning torque errors had two hidden layers, three hidden layer neurons and $\boldsymbol{q}$, ${}^{b}\boldsymbol{\omega}$, ${}^{b}\boldsymbol{\alpha}$ and $\boldsymbol{u}$ as inputs. The RMS error in the torque model was reduced to 0.98% of the nominal model, when the disturbance was added. These results were generated as a ratio of the RMS error between the model with disturbances and the basic dynamic model. Although the torque error was reduced to a significant amount, this model uses angular acceleration input. If the learnt unmodelled torque is used in a controller, a stability analysis must be carried out to avoid instabilities.

## 6.3 Research Summery of Objective III

Having a more accurate dynamic model, the last objective was set to demonstrate the performance improvement of the quadcopter when the compound dynamic model was

used in the controller. This was carried out by having a trajectory tracking controller in a quadcopter simulation. An LQR controller, which is a model-based controller was used as the high-level controller. The trajectory error was compared between with and without the learnt disturbance force model in the dynamic model of the controller. According to the results in Chapter 5, the trajectory error was reduced while using the compound dynamic model than using the basic model. Hence, it is concluded that the performance of the quadrotor trajectory tracking was improved by adding the learnt unmodelled components to the dynamic model.

## 6.4 Contributions

This research could achieve the following contributions.

- Detailed comparison of GPR and ANN to learn unmodelled forces of a quadcopter.
- Performance comparison of different ANN architectures.
- Extending the ANN approach to estimate the unmodelled torque components.

## 6.5 Future Directives

This research presented in this thesis can be extended to conduct experiments using an actual quadcopter. The work in this thesis is carried out assuming that all states are available from the measurements. This assumption may not valid in practical scenarios. Therefore, an observer can be modelled from the limited measurements. The learnt dynamics can be used in the observer to achieve more accuracy.

In addition, learning unmodelled quadrotor dynamics can be used in several applications where the performance depends on the dynamic model. Few of these applications include model-based control, payload manipulation and accurate trajectory tracking tasks.

# Bibliography


[1] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic mpc for model-based reinforcement learning," 2017, pp. 1714 – 1721.

[2] B. J. Emran, J. Dias, L. Seneviratne, and G. Cai, "Robust adaptive control design for quadcopter payload add and drop applications," in *2015 34th Chinese Control Conference (CCC)*, 2015, pp. 3252–3257.

[3] S. Islam, P. X. Liu, and A. El Saddik, "Robust control of four-rotor unmanned aerial vehicle with disturbance uncertainty," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 3, pp. 1563–1571, 2015.

[4] Q. Li, J. Qian, Z. Zhu, X. Bao, M. K. Helwa, and A. P. Schoellig, "Deep neural networks for improved, impromptu trajectory tracking of quadrotors," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5183–5189.

[5] Y. Wang, H. Zhang, and D. Han, "Neural network adaptive inverse model control method for quadrotor uav," in *2016 35th Chinese Control Conference (CCC)*, 2016, pp. 3653–3658.

[6] F. Berkenkamp and A. P. Schoellig, "Safe and robust learning control with gaussian processes," in *2015 European Control Conference (ECC)*, 2015, pp. 2496–2501.

[7] A. Kourani, K. Kassem, and N. Daher, "Coping with quadcopter payload variation via adaptive robust control," in *2018 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET)*, 2018, pp. 1–6.

[8] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S. Chung, "Neural lander: Stable drone landing control using learned dynamics," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 9784–9790.

[9] C. D. McKinnon and A. P. Schoellig, "Learn fast, forget slow: Safe predictive learning control for systems with unknown and changing dynamics performing

repetitive tasks," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2180–2187, 2019.

[10] Y. Liu, J. M. Montenbruck, P. Stegagno, F. Allgöwer, and A. Zell, "A robust nonlinear controller for nontrivial quadrotor maneuvers: Approach and verification," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 5410–5416.

[11] A. S. Sanca, P. J. Alsina, and J. d. J. F. Cerqueira, "Dynamic modelling of a quadrotor aerial vehicle with nonlinear inputs," in *2008 IEEE Latin American Robotic Symposium*, 2008, pp. 143–148.

[12] P. Kantue and J. O. Pedro, "Nonlinear identification of an unmanned quadcopter rotor dynamics using rbf neural networks," in *2018 22nd International Conference on System Theory, Control and Computing (ICSTCC)*, 2018, pp. 292–298.

[13] F. Jiang, F. Pourpanah, and Q. Hao, "Design, implementation, and evaluation of a neural-network-based quadcopter uav system," *IEEE Transactions on Industrial Electronics*, vol. 67, no. 3, pp. 2076–2085, 2020.

[14] B. Min, J. Hong, and E. T. Matson, "Adaptive robust control (arc) for an altitude control of a quadrotor type uav carrying an unknown payloads," in *2011 11th International Conference on Control, Automation and Systems*, 2011, pp. 1147–1151.

[15] S. Zhou, M. K. Helwa, and A. P. Schoellig, "Design of deep neural networks as add-on blocks for improving impromptu trajectory tracking," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, 2017, pp. 5201–5207.

[16] A. Aswani, H. Gonzalez, S. S. Sastry, and C. Tomlin, "Provably safe and robust learning-based model predictive control," *Automatica*, vol. 49, no. 5, pp. 1216 – 1226, 2013.

[17] A. Sarabakha and E. Kayacan, "Online deep learning for improved trajectory tracking of unmanned aerial vehicles using expert knowledge," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 7727–7733.

[18] S. Bansal, A. K. Akametalu, F. J. Jiang, F. Laine, and C. J. Tomlin, "Learning quadrotor dynamics using neural network for flight control," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, 2016, pp. 4653–4660.

[19] L. Wang, E. A. Theodorou, and M. Egerstedt, "Safe learning of quadrotor dynamics using barrier certificates," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 2460–2465.

[20] A. K. Akametalu, J. F. Fisac, J. H. Gillula, S. Kaynama, M. N. Zeilinger, and C. J. Tomlin, "Reachability-based safe learning with gaussian processes," in *53rd IEEE Conference on Decision and Control*, 2014, pp. 1424–1431.

[21] A. Punjani and P. Abbeel, "Deep learning helicopter dynamics models," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 3223–3230.

[22] A. I. Galushkin, *Neural network theory*, 2007. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-48125-6

[23] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85 – 117, 2015.

[24] N. Mohajerin and S. L. Waslander, "Modular deep recurrent neural network: Application to quadrotors," in *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2014, pp. 1374–1379.

[25] M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. Pappas, "Efficient and accurate estimation of lipschitz constants for deep neural networks," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 11 427–11 438.

[26] J. Franke and M. Neumann, *Bootstrapping neural networks*. Technische Universita¨t Kaiserslautern, Fachbereich Mathematik, 1998.

[27] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the marquardt algorithm," *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 989–993, 1994.

[28] C. Rasmussen and C. Williams, *Gaussian processes for machine learning*. Mass.: MIT Press, 2006.

[29] F. Berkenkamp, A. P. Schoellig, and A. Krause, "Safe controller optimization for quadrotors with gaussian processes," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 491–496.

[30] W. Neeley, "Design and development of a high-performance quadrotor control architecture based on feedback linearization," 2016. [Online]. Available: https://digitalrepository.unm.edu/ece_etds/190

[31] R. Faragher, "Understanding the basis of the kalman filter via a simple and intuitive derivation [lecture notes]," *IEEE Signal Processing Magazine*, vol. 29, no. 5, pp. 128–132, 2012.

[32] J. Solà, "Quaternion kinematics for the error-state kalman filter," *CoRR*, vol. abs/1711.02508, 2017. [Online]. Available: http://arxiv.org/abs/1711.02508

[33] J. D. J. Solà and D. Atchuthan, "A micro lie theory for state estimation in robotics," *arXiv:1812.01537 [cs]*, 2018.