Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Systems Lab
Prof. Roland Siegwart

Semester Thesis

Online Neural Network Based Model Identification of a Fixed-Wing UAV

Spring Term 2024

Supervised by:
Thomas Stastny
Florian Achermann

Author:
Marko Maljkovic

Contents

Acknowledgments

I would like to thank my supervisors, Thomas Stastny and Florian Achermann for their advice, patience and support throughout the course of the project. Furthermore, I would like to thank David Rohr for helping me gather all the data used in this semester project.

Zurich, 2 September 2019

Marko Maljkovic

Abstract

In order to design controllers that achieve specific performance criteria, one needs to know an adequate model of the system. For Fixed-Wing UAVs, the applications that require attitude, altitude or velocity reference tracking inherently require knowledge of the UAV's dynamic model. We focus on the longitudinal dynamics model of the system and for a Fixed-Wing UAV it is particularly hard to design such a model without previously flying the plane. The reason for this lies in the fact that the aerodynamic forces and moments that act on the plane are largely dependant on the flying conditions and aircraft configuration and can only be known once the plane is deployed. Therefore, in this thesis we aim to design an automated, online procedure that learns while the plane is flying. The goal is to bypass the standard frequency domain identification procedures and try to learn a time domain mapping from specifically chosen input variables to the outputs of our interest. Since Neural Networks can learn nonlinear functions from sufficient amount of data and with little to no a priori given information about the structure of the function, we try to modify this approach by using a finite length, time domain, sliding window for data gathering so as to fulfill the real time, online learning requirements. As we are interested in the longitudinal dynamics, we try to map the angle of attack α, air speed V, pitch rate q and elevator deflection δ_E to the linear acceleration in the $x-$ axis, $z-$ axis and $\dot{q}$. We propose three Neural Network based model structures and a parameter space over which we will demonstrate the effects imposed by choosing a particular combination. To evaluate the performance of each model, we perform real time simulations where we train the models for different parameter choices and for different training trajectories and report the obtained results. Using the Grid Search procedure, we find that the combination of model structure and hyperparameters that achieves the best combined estimation in all three output channels is a Neural Network with two hidden layers, each with 40 hidden neurons, with window size of $N_W = 200$ and learning rate $\eta = 0.001$. For this model we estimate the convergence time of the training loss curve to be 30 sec and we give certain remarks about the influence of the inputs used during training on the performance of the fully trained model.

Symbols

Symbols

ϕ, θ, ψ	roll, pitch and yaw angle

Indices

x	x axis
y	y axis

Acronyms and Abbreviations

ETH	Eidgenössische Technische Hochschule
IMU	Inertial Measurement Unit
UAV	Unmanned Aerial Vehicle
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
ANN	Artificial Neural Network
RNN	Recurrent Neural Network
CPU	Central Processing Unit
ROS	Robotic Operating System
HIL	Hardware in the loop
NED	North East Down reference frame
ENU	East North Up reference frame
FRD	Forward Right Down reference frame
SGD	Stochastic Gradient Descend

Chapter 1

Introduction

1.1 Motivation

The small fixed-wing Unmanned Aerial Vehicles (UAVs) have over the years become a very popular framework for developing and testing novel flight control systems and guidance algorithms. As a platform, these vehicles are attractive due to their low production cost, high reconfigurability and limited risk of harm in the event of failure. Since they can be remotely controlled or autonomous they can be used for the surveillance [1], [2], search and rescue missions [3], [4], agricultural application [5] etc.
Essentially, for every UAV application we have to solve two problems:

1. Develop a dynamics model that captures all the physical effects relevant for the particular application

2. Design the control algorithm that utilizes previously obtained model (MPC for linearized systems, Nonlinear MPC etc.)

First problem can be tackled in either frequency [6], [7] or time domain [8], [9], [10], [11], [12]. If done in the frequency domain, the problem becomes one of parameter identification of an a priori chosen transfer function from command inputs (for example throttle, aileron, elevator and rudder deflections) to system states - usually pose, linear and angular velocities and accelerations. In time domain, we either perform one or multiple steps ahead in time prediction of the system's states or we use Machine Learning based parameter identification techniques to establish time domain system models. In both cases, the achieved accuracy undoubtedly dictates the limitations of any control algorithm that is to be implemented based upon the obtained dynamics model. The underlying complex and coupled relations between fixed-wing UAV's pose, airflow angles, linear and angular rates, aerodynamic forces and moments are given by the Newton-Euler equations [13]. Their highly nonlinear nature imposes a limited degree of approximation capability to any model linearized around an operating point. With that in mind, attempts are constantly made to find better-suited, easy to implement, nonlinear approximators, independent of the vendor and application specific plane configurations (different sizes, air frame constructions, sensor placement etc.) that capture all the important dynamic effects.
The second problem deals with the stability and performance issues of the system when in use for a particular purpose. Control Theory and Convex/Non Convex optimization techniques are the tools that allow real time control. However, the second problem is beyond the scope of this project and will not be addressed in following chapters.

1.2 Problem statement

In this thesis we focus only on the system identification problem and we investigate the potential use of an Artificial Neural Network based approach for online learning of the fixed-wing UAV's longitudinal dynamics model. We want to allow the model to learn while the plane is flying. Ideally, we try to replace all the platform specific tuning with a short and universal model identification training procedure that is completely independent of the aircraft being used. Given the size and the computational power of the available on-board computers, it is crucial to design identification methods that do not require high memory and CPU usage but at the same time provide required level of accuracy, crucial for the real time control applications. Therefore, using the ANNs is by all means a reasonable approach as they are adaptive models that can treat the unknown and changing dynamics as a black-box that maps the adequately chosen in-flight measurements to desired system states. In fact, ANNs learn through examples and can be used as a function approximation. In the end, the aim is to create computationally efficient, simple, yet accurate enough models that allow users to train the system in real time by performing short random flight maneuvers with the plane. The main goals can be formulated as follows:

- Design and implementation of several Neural Network based models in PyTorch. The models should capture the nonlinear longitudinal dynamic effects introduced through the Lift (C_L), Drag (C_D) and Pitching Momentum (C_m) coefficients in Newton - Euler's equations. The outputs that are to be identified are linear accelerations in the body x-axis denoted by a_x, the body z-axis, denoted by a_z and the time derivative of the pitch rate, denoted by $\dot{q}$. Since we will be using a time domain, data window approach, the hyperparameters whose influence on the model performance will be analyzed are given by: learning rate η_t, number of neurons in hidden layers and window size N_W.
- Training the models in a ROS simulated online setup with five different training data sets obtained through Hardware in the Loop (HIL) simulations. The performance of the trained models will than be cross validated and the optimal one will be chosen by Grid Search algorithm.
- Analyzing the average convergence time of the training loss function over the set of different training trajectories for the model that achieves the best performance. Furthermore, we look at some of the effects that different types of training input trajectories can have on the learning procedure by examining the validation loss curves of the optimal model.

1.3 Structure of the thesis

The schematic representation of the stages in the project is given in Figure 1.1

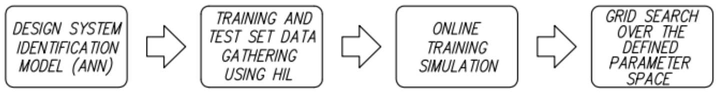
DESIGN SYSTEM IDENTIFICATION MODEL (ANN)
TRAINING AND TEST SET DATA GATHERING USING HIL
ONLINE TRAINING SIMULATION
GRID SEARCH OVER THE DEFINED PARAMETER SPACE

Figure 1.1: Schematic representation of the various stages in the project

In chapter 2, we first define the model of the longitudinal dynamics to be identified and review the literature to investigate different approaches and model structures that have already been used. Based on the documented attempts, in chapter 3 we define three promising model architectures and establish the reasonable range for the values of the model parameters. Chapter 4 defines the online setup for training the models. We explain the design of the essential parts of the project - data logging and data preparation procedure and resolve some practical implementation issues regarding the simulation that have to be addressed beforehand. Finally, we present the data sets that are used for training and validation of the models. In chapter 5 we evaluate each of the models over the space of chosen model parameters and perform a grid search in order to find the model that best generalizes the longitudinal dynamics on the available data sets. Afterwards, we discuss the results of the training and validation procedure and analyze the RMSE achieved for different combination of model structures, parameters, training and validation sets. We also discuss some of the observed properties of the training trajectories.

Chapter 2

Longitudinal Dynamics of a small, fixed-wing UAV

2.1 Euler equations

The complete nonlinear dynamics model of the fixed-wing UAV, expressed in the body frame, is given by the following Newton-Euler equation [13]:

$$\begin{bmatrix} mE_{3x3} & 0 \\ 0 & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} {}_B\dot{\boldsymbol{v}} \\ {}_B\dot{\boldsymbol{\omega}} \end{bmatrix} + \begin{bmatrix} {}_B\boldsymbol{\omega} \times m {}_B\boldsymbol{v} \\ {}_B\boldsymbol{\omega} \times \boldsymbol{I} {}_B\boldsymbol{\omega} \end{bmatrix} = \begin{bmatrix} {}_B\boldsymbol{F} \\ {}_B\boldsymbol{M} \end{bmatrix} \tag{2.1}$$

where $E_{3\times 3}$ is the Identity matrix, $\boldsymbol{I}$ is the Inertia matrix, ${}_B\boldsymbol{v}$ is the liner velocity vector expressed in the body frame, ${}_B\omega$ is the angular velocity vector expressed in the body frame, and ${}_BF$ and ${}_BM$ are the resulting force and moment that act on the aircraft, expressed in the body frame.

Model (2.1) has six coupled, nonlinear equations but for the purpose of this thesis it suffices to work only with three of these six equations. As we are only interested in the longitudinal dynamics of the system, we only need to look at the force balance equations in the x-axis and z-axis and the moment balance for rotation around the y-axis. The resulting model is given by equations:

$$m\dot{u} = ma_x = F_T \cos\varepsilon - D\cos\alpha + L\sin\alpha - mg\sin\theta + mrv - mqw \tag{2.2}$$

$$m\dot{w} = ma_z = F_T \sin\varepsilon - D\sin\alpha - L\cos\alpha + mg\cos\phi\cos\theta + mqu - mpv \tag{2.3}$$

$$I_{yy}\dot{q} = M_m + M_{m_T} - pr\left(I_{xx} - I_{zz}\right) - \left(r^2 - p^2\right) I_{xz} \tag{2.4}$$

where u and p denote the x-axis linear and angular velocity, q denotes the y-axis angular velocity, w and r denote the z-axis linear and angular velocity, F_T is the thrust force that acts under a certain angle ϵ, θ, ϕ and α represent pitch angle, roll angle and angle of attack, g is the gravitational acceleration, L and D are the resulting lift and drag forces and M_m and M_{m_T} are the pitching and thrust moments that act around y-axis. The forces relevant for the longitudinal dynamics are presented in Figure 2.1.

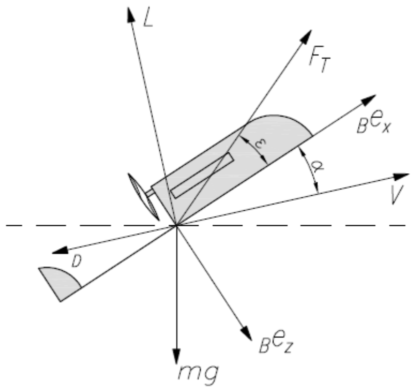

Figure 2.1: Forces acting on the UAV

In this problem setup, we use the data only from gliding maneuvers of the fixed-wing UAV which implies $F_T = 0$ at all times. If we were to use the throttle as well, we would in fact try to identify a system which would be a combination of the aerodynamics model of the plane and the dynamics of the used aircraft engine. The installed IMU can only measure the total acceleration, which would be the combined result of thrust and aerodynamic forces acting on the UAV. This means that there exists no way to determine the individual contributions of the components. Thus, in all the data gathering experiments, no thrust is being used in order to ensure that the information encoded in the data captures only the aerodynamic effects of the aircraft as this is what we aim to identify. Furthermore, the contribution of the Coriolis acceleration in the equations (2.2) to (2.4) is negligible and therefore could also be neglected so as to additionally simplify the model to the form:

$$ma_x = -D\cos\alpha + L\sin\alpha - mg\sin\theta \tag{2.5}$$

$$ma_z = -D\sin\alpha - L\cos\alpha + mg\cos\phi\cos\theta \tag{2.6}$$

$$I_{yy}\dot{q} = M_m - pr\left(I_{xx} - I_{zz}\right) \tag{2.7}$$

If we re-arrange the equations (2.5) and (2.6) by moving the gravitational terms to the left hand side of the equality we have:

$$a_x^{meas} = a_x + g\sin\theta = \frac{1}{m}\left(-D\cos\alpha + L\sin\alpha\right) \tag{2.8}$$

$$a_z^{meas} = a_z - g\cos\phi\cos\theta = \frac{1}{m}\left(-D\sin\alpha - L\cos\alpha\right) \tag{2.9}$$

where a_x^{meas} and a_z^{meas} are the values we have available from the aircraft's IMU. The complete model of the forces acting on the aircraft is given in Figure 2.1.
In the resulting model, the Lift force, Drag force and Pitching moment are given by equations:

$$L = \frac{1}{2}\rho V^2 S C_L \tag{2.10}$$

$$D = \frac{1}{2}\rho V^2 S C_D \tag{2.11}$$

$$M_m = \frac{1}{2}\rho V^2 S\bar{c} c_m \tag{2.12}$$

where ρ is the air density, V is the free stream velocity, S is the reference area, $\bar{c}$ is the mean geometric chord and C_L, C_D and c_m are the lift, drag and pitch moment coefficients that are highly nonlinear, unknown functions of the angle of attack. Each of them can be expressed as:

$$C_L = f_{C_L}(\alpha) \tag{2.13}$$

$$C_D = f_{C_D}(\alpha) \tag{2.14}$$

$$c_m = f_{c_m}(\alpha) \tag{2.15}$$

By jointly looking at the equations (2.7) to (2.15), it is clear that the fundamental concept of this approach is to use the ANNs to account for the three unknown functions f_{C_L}, f_{C_D} and f_{c_m} when estimating the linear accelerations and $\dot{q}$. In the next section, we do a literature review of the reported successful Neural Network based architectures used in the past for aircraft parameter identification purposes.

2.2 Literature review of the ANN based models

We are interested in developing as simple of a model as possible that needs somewhat short training procedure, conducted preferably with random maneuvers. Qian, Nadri, Morosan, and Dufour [14] introduce a method for closed loop on-line parameter identification of dynamic systems by designing a state observer that considers the unknown parameters as the entries in the state vector. This is now used in an MPC-like technique for calculating the reference signals so as to maximize the information contained in the system input according to the Fischer information matrix. Morelli [15] proposes a method for online optimal input design for state and control derivative parameter identification. In his paper he explains the procedure for linearized systems but claims it can be just as easily adapted for the nonlinear identification. These and similar procedures are computationally expensive in themselves and as such violate the idea of having a simple but efficient training procedure. Garcia and Keshmiri [8] present a simple Neural Network structure with one hidden layer and 10 neurons that utilizes a sliding window in time domain to form a training set for each epoch. They report using the tangent hyperbolic activation function for the hidden layer neurons, a window size of 10 and a sampling frequency of $f = 10$ Hz. Similarly, Samal, Anavatti, and Garratt [9] and Peyada and Ghosh [10] in their papers use similar Neural Network structures with just one hidden layer and 10 to 20 neurons with tangent hyperbolic activation function, with the second paper reporting the data sampling frequency of $f = 50$ Hz. Puttige [11] and Zhang [12] propose Recurrent Neural Network structures for one step ahead in time prediction of the system state. Inherently, these structures represent auto-regressive exogenous input models and are more suitable for control systems design. Since we attempt to identify the underlying low-level dynamics model of the aircraft, it makes more sense to choose a non RNN model for this purpose. Regarding the chosen input signals for the training procedure, authors have used different types. Some of the suggested include:

- Random signals: designed to include all the flight regimes and modes of the aircraft [11]
- Random binary sequence: signals that can take just two different values in a randomly chosen manner [12]
- Multi step elevator input signal [10]
- Sine wave, frequency sweep signals, singlets and doublets [16]
- any combination of the above mentioned signals

In the next chapter we will present the three chosen Neural Network based model structure whose performance will be analyzed.

Chapter 3

ANN models

3.1 Outputs of the system

In the previous section, we reviewed successful experimental setups used for Neural Network based dynamics model identification. In this thesis we propose and analyze three models with similar complexity (in terms of number of tunable parameters) but different architectures. Firstly, we define the set of outputs of our system. As suggested by equations (2.10) to (2.12), we could train the Deep Learning model to directly map appropriately chosen inputs to the Lift, Drag and Pitching moment coefficients. However, it might be more practical and robust to estimate values that can be directly measured - the IMU measurements a_x^{meas} and a_z^{meas}. This is important for all the measurements that we can obtain are effected by noise. Neural Networks use back propagation to learn, which is in essence, just calculating the derivatives of the chosen cost function with respect to the model parameters. Given how sensitive to noise derivatives are, it might be the case that back propagating the error between some nonlinear combination of the noise corrupted measurements hand-labeled as the true value, and the value estimated by the network can lead to bad results, slow convergence or, in severe cases, model divergence. As mentioned before, if we impose restriction to gather data just in the gliding mode, than we can directly measure a_x and a_z. Having these two values allows us to directly estimate the Lift and Drag coefficients if needed as:

$$C_L = f_{C_L}\left(\text{model inputs}, \text{model outputs}\right) \tag{3.1}$$

$$C_D = f_{C_D}\left(\text{model inputs}, \text{model outputs}\right) \tag{3.2}$$

Unfortunately, we cannot directly measure the angular acceleration and therefore, in order to estimate c_m, we have to first approximate $\dot{q}$. Approximating the derivative of a measurement is in every sense as sensitive to noise as the back propagation procedure itself. To better the robustness with respect to the measurement noise, we use the central difference approximation of the first derivative given by:

$$\dot{q_k} \approx \frac{q_{k+1} - q_{k-1}}{2\mathit{\Delta} T} \tag{3.3}$$

where q_k, q_{k-1} and q_{k+1} denote the pitch rate at respective time instances and $\mathit{\Delta} T$ corresponds to the sampling period of the system. Keeping everything above mentioned in mind, we choose the outputs of our Neural Network based model to be:

1. the total x-axis linear acceleration of the aircraft expressed in the body frame a_x^{meas}. From this point on, we denote this output by a_x

2. the total z-axis linear acceleration of the aircraft expressed in the body frame a_z^{meas}. From this point on, we denote the second output of the system by a_z
3. y-axis angular acceleration $\dot{q}$

For the first two outputs we have direct measurements to compare to, whereas for the $\dot{q}$ measurement we have to hand label the input data points to the appropriate central difference approximation. Note that this approximation is not causal in its nature and as such can only be used on the previously gathered data.

3.2 Inputs to the system

Next thing we have to define is the minimal set of inputs we want to use that is capable of capturing the nonlinear nature of the system. By looking at the Euler-Newton's equations we can infer the inputs that are essential for the training procedure. In terms of exogenous inputs, only elevator deflection has an impact on the longitudinal model of the UAV. Given that most of the unmodeled nonlinearities are contained in the Lift, Drag and Pitching Moment coefficient's dependency on the angle of attack, we ought to use α as one of the inputs to the system. In addition, pitch rate q and free stream velocity V also have an influence on the longitudinal dynamics and as such will be included in the system inputs as well. The resulting set of inputs for all models is given by:

1. Angle of attack α
2. Pitch rate q
3. Free stream velocity V
4. Elevator deflection δ_e

3.3 Neural Network based models

In typical Data Science Deep Learning regression problems, we usually form a model with one or more fully connected layers that estimates all the values of interest at once. In a similar manner we design our model as well. However, given the problems imposed by the noise corrupted measurement data, we make a design choice to put the pitch rate derivative estimation in a separate branch of the Neural Network. That way, the risk of back propagating the noise errors created in the label designing process is reduced to minimum.

3.3.1 Model 1

The first model we will use has the most simple structure of the three presented in this thesis. It is a two branched Neural Network with four inputs, two outputs in the first branch and one output in the second branch. Each branch has one hidden layer fully connected to the input layer and fully connected to the output layer. The activation function of each hidden neuron is tangent hyperbolic function:

$$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3.4}$$

whereas the nodes in the output layer have linear activation function. Thus, the output of each hidden neuron is given by:

$$F_j = \phi\left(\sum_{i=1}^{4} w_{ji}x_i + b_j\right) \tag{3.5}$$

where w_{ji} and b_j are the corresponding weights and biases to be learned through training. The structure of the model is presented in Figure 3.1

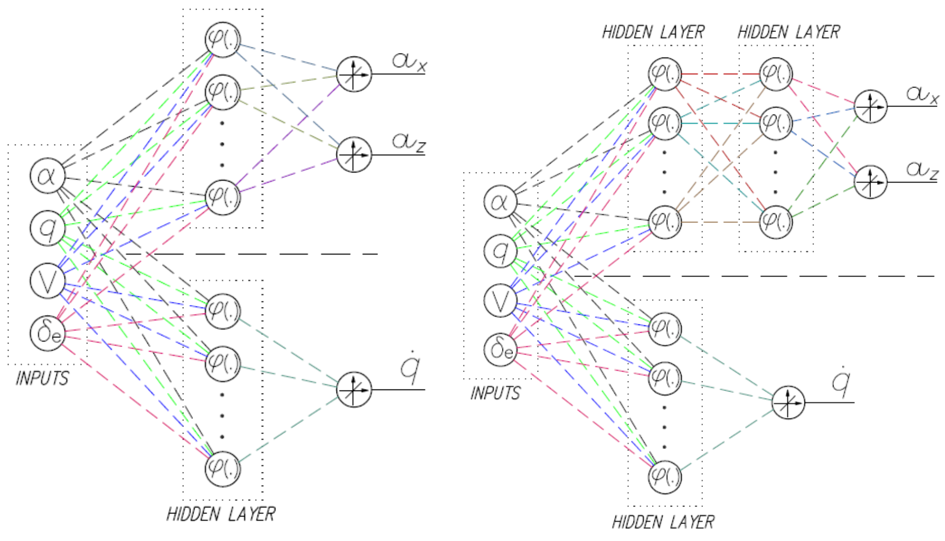

Figure 3.1: Model 1 on the left and Model 2 on the right

3.3.2 Model 2

Second model to be used in this project is depicted in Figure 3.1 on the right. This model differs from the first one in the fact that it has an extra hidden layer in the first branch. Since extra model parameters imply better learning capabilities but also impose longer training times and slower convergence, we have to find a compromise between the accuracy requirements and the real time application requirements. Since two hidden layers already achieve good estimation accuracy, we choose not to test the models with larger number of hidden layers. Furthermore, the real time computation capabilities of the system can be violated if we use more than two hidden layers as extra nodes dictate longer training epochs. We keep the same activation functions for the second hidden layer and set the number of hidden nodes to be equal to the number of nodes in the first hidden layer.

3.3.3 Loss functions and optimizer

Loss function for accelerations

For the training procedure, we also have to define the loss functions with respect to which we will calculate the gradients and update the model weights. Typically, a MSE loss function is used. For the purpose of coupled identification of the x-axis and z-axis linear accelerations, it is important to note that the order of magnitude of these outputs differs. In a random flight maneuver, a_z can be more than ten times larger than the corresponding a_x acceleration. This means that the gradient based update procedure would focus more on reducing the estimation error of a_z and therefore would somewhat neglect the contribution of the a_x error to the combined loss function. To prevent this from happening, before we use the measured values of a_z acceleration as labels, we re-scale them in the following way:

$$\hat{a}_z = \frac{a_z + 9.81}{10} \tag{3.6}$$

Note that a_z measured by the IMU should be a negative number if represented in the NED reference frame. However, the Mavros topic that we will use gives us the a_z value expressed in the forward-left-up coordinate system and we have to subtract 9.81 in our scaling procedure instead. With the scaled outputs of the system we can now use the combined MSE loss function given in the form:

$$L\left(a_x, \hat{a}_x, a_z, \hat{a}_z\right) = \frac{1}{|D|}\sum_{i=1}^{|D|}\left(a_{x,i} - \hat{a}_{x,i}\right)^2 + \left(a_{z,i} - \hat{a}_{z,i}\right)^2 \tag{3.7}$$

where $|D|$ is the number of samples in the batch, $a_{x,i}$, $a_{z,i}$ are the measured accelerations of the i-th training sample and $\hat{a}_{x,i}$ and $\hat{a}_{z,i}$ are the estimated accelerations. Note that the scaling parameters are a priori determined fixed constants. These constants are by no means optimal but since we do not have any information about the available value ranges of the input signals before we actually fly the plane, there is no way to find some more appropriate scaling parameters.

Loss function for $\dot{q}$

Since there is no direct measurement of the pitch rate derivative we have to define a custom loss function for this estimation. Although we design the labels for the training samples in the data preparation step of the algorithm, these labels are for sure corrupted by noise. Therefore, we introduce an additional term in the loss function which requires that the forward integration of the previous $\dot{q}$ estimate over a time period of one sampling period be equal to the measured value of q in the current discrete time instance. The contribution of a single training sample to the cumulative loss of the whole training batch is given by:

$$l_i\left(\hat{\dot{q}}_i, \hat{\dot{q}}_{i-1}, \dot{q}_i, q_{i-1}, q_i\right) = \left(\dot{q}_i - \hat{\dot{q}}_i\right)^2 + \left(q_i - q_{i-1} - \hat{\dot{q}}_{i-1}\Delta t\right)^2 \tag{3.8}$$

where $\hat{\dot{q}}_i$ and $\hat{\dot{q}}_{i-1}$ represent the estimated values at respective discrete time instances, q_i and q_{i-1} are the measured values at respective discrete time instances, $\dot{q}_i$ is the designed label for the current sampling point and Δt is the sampling period. The loss function defined for the whole training batch is now defined as:

$$L = \sqrt{\frac{1}{|D|}\sum_{i=1}^{|D|} l_i\left(\hat{\dot{q}}_i, \hat{\dot{q}}_{i-1}, \dot{q}_i, q_{i-1}, q_i\right)} \tag{3.9}$$

Optimizer

The final part of designing the training procedure is choosing the optimizer for the weight update. As we are designing a time critical, online training procedure we decide to use a modified version of Stochastic Gradient Descend called Adam that reports faster convergence times in literature [17]. Often times the default values of moving average parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$ have proven to work excellent. In this thesis we use the default values. Optimization over the set of different values for β_1 and β_2 is not covered in this thesis and should be discussed in future work related to this topic.

3.3.4 Model 3

In the previous subsection we explained the reasons for scaling the a_z output of the system and gave the joint MSE loss function for the back propagation of the linear accelerations. With this model we bypass the need for scaling and try to achieve better accuracy by placing each of the estimation outputs in a separate branch of the Neural Network. We also aim to investigate if some of the performance is perhaps lost due to no longer imposing a coupled nature of the acceleration outputs. Apart from separating the branches, no change is made compared to the Model 1 and Model 2. The activation functions of all the hidden neurons stay the same and all the output neurons remain with the linear activation. As this three branch model introduces higher complexity to begin with, we make no attempt to analyze the more complex models that would introduce multiple hidden layers in some of the branches. The resulting structure of the third model is presented in Figure 3.2.

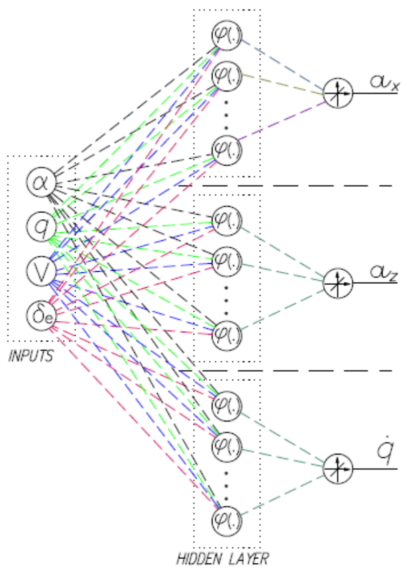

Figure 3.2: Model 3

3.4 Final improvements

One major problem that could happen in the online learning setup is the problem of overfitting. Since we try to avoid using predefined validation sets so as to allow system to learn and predict adaptively, we cannot use standard techniques such as early stopping to prevent it. We opt for a solution which is typical for systems with large amount of hidden neurons and hidden layers - dropout. It essentially introduces a probability that a certain node will be included in the forward and back propagation procedure and by randomly turning off the nodes in hidden layers it tries to prevent the system from overfitting. As we have no probabilistic interpretation we impose the default value of dropout probability $p = 0.5$. As we are using the tanh activation functions in hidden layers we can use the Xavier Initialization of the weights. Initializing the tanh hidden layers like this should somewhat help our system with the common Machine Learning problem of exploding or vanishing gradients.

Chapter 4

Online training procedure

In this chapter we describe the Online training setup used for training and performance evaluation of the three models described in the previous section. All the simulations are performed on Ubuntu 18.4 64bit Virtual Machine with two processor cores and 4GB of combined RAM and graphics memory. The simulations are done using ROS Melodic (version 1.14.3) and PyTorch. One node is used to play previously recorded .bag files containing all the topics required to describe the training trajectories. A listener node is written in Python and is designed so as to simultaneously listen to subscribed topics, collect and store all the necessary ROS messages, interpolate messages so that a constant sampling time for training data samples is imposed, prepare training sets for each training epoch and perform the online update of the model parameters. The entire online procedure can be described as a process that goes through 3 Modules:

- Module 1: ROS topic message logging module
- Module 2: ROS message interpolation module used to re-sample the messages
- Module 3: Time domain sliding window module that prepares the training set for current iteration

4.1 Module 1: Message logger

Firstly, we design a Data Buffer class to be used for storing the incoming messages from individual ROS topics. This represents the Module 1 of the training procedure. To limit the required storage space, we impose the maximum number of stored messages for each individual topic. Now, when the message set of a certain topic is full and a new message arrives, the oldest message should be removed. This makes the Python deque structure a perfect solution as it performs the push and pop operations on the first and last element in $o(1)$ time complexity - deque structure represents a connected list with a pointer to the first and the last element of the list. One thing worth noting is the fact that the stored messages do not contain all the required inputs for our model. We can directly subscribe to ROS topics that publish pitch rate, elevator deflection, a_x and a_z but in order to get the angle of attack and free stream velocity we have to subscribe to the topics that give as aircraft's orientation, ground velocity and estimate of the wind velocity. All the ROS topics used in the scope of this thesis are presented in the table 4.1. When we have a module that collects all the necessary ROS messages we proceed to define the second module that tackles the time synchronization of the received measurements problem and explains how the angle of attack and free stream velocity are calculated as the training inputs.

Table 4.1: Subscribed ROS topics

ROS topic	Description
/mavros/global_position/local	receives ground velocity
/mavros/wind_estimation	receives wind velocity estimate
/mavros/imu/data_raw	receives a_x and a_z and angular rates
/mavros/local_position/pose	receives UAV orientation
/mavros/imu/static(diff)_pressure	receives static(differnetial) pressure
/mavros/imu/temperature_imu	receives the temperature
/mavros/target_actuator_control	receives elevator deflection

4.2 Module 2: Data Interpolation

Now that we have all the data needed for creating the inputs, we need to transform them in the appropriate form. In a real world scenario, various sensor measurements are gathered and sent to the on board computer for real time processing. Since the measurements can come from different devices it is very likely that the average frequency of their arrival differs. This imposes the first problem denoted as data synchronization. Second module that performs the ROS message interpolation at equally spaced time instances prepares the adequate training sample points. The schematic diagram of the proposed solution is given in Figure 4.1

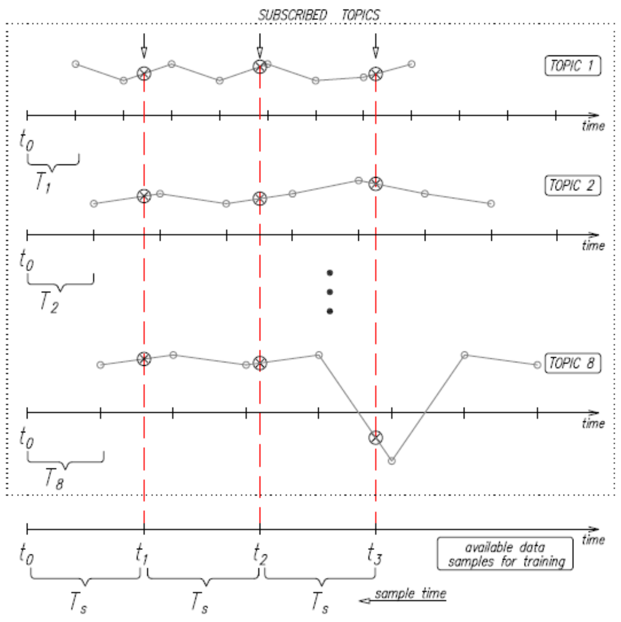

Figure 4.1: Schematic representation of data gathering procedure

Let us define T_s to be the sampling period of the Neural Network based models. In literature, the reported values of T_s are in the range from 0.01 sec to 0.1 sec. In order to capture the fast changes in the system inputs we set the sampling period to be equal $T_s = 0.05$ sec. That way, we restrict ourselves to using only the training samples that are equally spaced in time. Consequently, the subscribed topics usually do not provide direct measurements at the discrete time instances $t_k = kT_s$ and require some kind of time domain data interpolation. Given that the time interval between two successive message arrivals is short, it makes sense to try just the linear interpolation between the two closest received messages as presented in the figure 4.1. The same interpolation procedure is applied for each of the subscribed topics. Finally, we form a new deque of size N_W where each entry is a set of transformations applied to the interpolated messages and corresponds to one training sample.

4.2.1 Forming a training sample

This section describes the transformations that are to be performed on the interpolated messages received from the ROS topics listed in table 4.1. Given that we have direct measurements for a_x, a_z, q and δ_e, we need to calculate the angle of attack and free stream velocity in order to have a complete training sample at each $t_k = kT_s$ discrete time instance. To obtain α and V we first subtract the estimated wind velocity from the given UAV ground velocity to get the air relative velocity expressed in the inertial frame. Here, we have to be careful about two things:

- For all the calculations we need the velocities to be expressed in the body NED frame. Mavlink does send the data expressed in NED frame but on the Mavros side the data is usually transformed into the ENU frame. By examining the plugin code for /mavros/global_position/local topic, we discover that this topic publishes the ground velocity in the form $({}_I u_g, {}_I v_g, -{}_I w_g)$ where ${}_I u_g$, ${}_I v_g$ and ${}_I w_g$ correspond to the components expressed in the ENU frame. Therefore, we have to change sign of the z-axis component before we continue.
- The /mavros/wind_estimation topic only publishes the wind velocity estimates in the xy plane. Therefore, we are forced to neglect the wind velocity component present in the z-axis.

The interpolated orientation quaternion $\epsilon = (\epsilon_0, \epsilon_1, \epsilon_2, \epsilon_3)$ derived from the corresponding ROS topic describes the orientation of the base link frame with respect to the ENU frame. By computing the adequate rotation matrix from the given quaternion and multiplying it with the aircraft velocity vector expressed in the inertial frame, we get the velocity vector expressed in the body link frame [18]. To express the velocity in the FRD body frame we change the sign of the y and z velocity components. The velocity components expressed in the body frame are given by:

$$ {}_B u = \left(\epsilon_0^2 + \epsilon_1^2 - \epsilon_2^2 - \epsilon_3^2\right) \ {}_I u_g + 2\left(\epsilon_1\epsilon_2 + \epsilon_0\epsilon_3\right) \ {}_I v_g + 2\left(\epsilon_1\epsilon_3 - \epsilon_0\epsilon_2\right) \ {}_I w_g \tag{4.1} $$

$$ {}_B v = -2\left(\epsilon_1\epsilon_2 - \epsilon_0\epsilon_3\right) \ {}_I u_g - \left(\epsilon_0^2 - \epsilon_1^2 + \epsilon_2^2 - \epsilon_3^2\right) \ {}_I v_g - 2\left(\epsilon_2\epsilon_3 + \epsilon_0\epsilon_1\right) \ {}_I w_g \tag{4.2} $$

$$ {}_B w = -2\left(\epsilon_1\epsilon_3 + \epsilon_0\epsilon_2\right) \ {}_I u_g - 2\left(\epsilon_2\epsilon_3 - \epsilon_0\epsilon_1\right) \ {}_I v_g - \left(\epsilon_0^2 - \epsilon_1^2 - \epsilon_2^2 + \epsilon_3^2\right) \ {}_I w_g \tag{4.3} $$

Free stream velocity and angle of attack are now given by:

$$ V = \sqrt{{}_B u^2 + \ {}_B v^2 + \ {}_B w^2} \tag{4.4} $$

$$ \alpha = \arctan\left(\frac{{}_B w}{{}_B u}\right) \tag{4.5} $$

4.3 Module 3: Training set creation

We explained the procedure for obtaining a single training sample and defined the data sampling period T_s in the previous section. Let us now define T_{update} to be the model update period - time interval between successive weight updates. Update period is chosen to be significantly larger than the sampling period so that both the data gathering and back propagation stages can fit into one update period. For practical reasons, we choose the update period to be $T_{update} = 5T_s = 0.25$ sec. Now, we implement a sliding window technique as described in [8]. A window is naturally imposed by the Data Buffer structure previously described. Since the Data Buffer keeps track of the last N_W (denotes the Buffer size) equally spaced in time data points, we aim to use currently stored samples as a training set for the current epoch. In this setup, N_W is a very important tuning parameter as it determines the time span in the past that is being used for model update. By choosing

$$N_W > \frac{T_{update}}{T_s} \tag{4.6}$$

we impose overlapping windows which means that some of the data points will be used in multiple training epochs. The schematic representation of the overlapping window structure is presented in Figure 5.1.

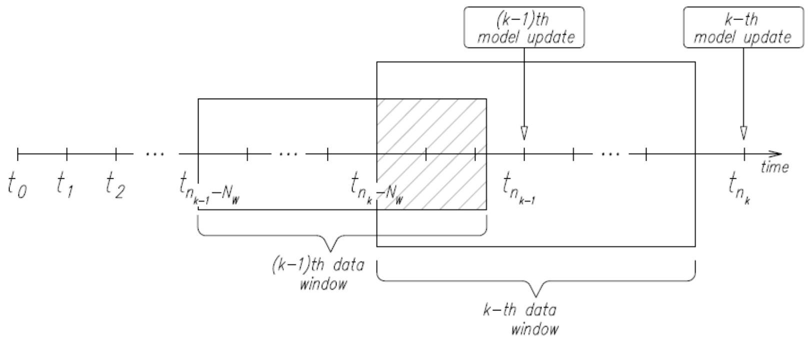

Figure 4.2: Schematic representation of the overlapping sliding window

Note that it is highly beneficial for the training procedure to see the same data points multiple times compared to using it just once and than discarding it. In a typical offline training setup, several update iterations are performed during one training epoch with the same batch of data. The overlapping window structure sort of mimics the same effect just in a more practical way as it is also using the newly arrived data points. Furthermore, when a multiple iterations per epoch approach was tested, the computational time required for back propagation increased significantly.

4.4 Parallelism

We implement the listener node so that it has a very important feature - Message Logging procedure and Model Update procedure with Data Preparation are done in parallel. This means that while the model is performing the back propagation, the Data Buffer is Simultaneously listening to all the new arriving messages. To be more precise, the only time the Message logging procedure is not conducted in parallel is when a certain data is requested from the memory. Then, we stop with the listening procedure for a short period of time while the data is being copied and as soon as it has been fetched from the memory, the operations continue in parallel. Since we are doing the interpolation procedure very often, this mechanism is important in order to prevent race conditions from happening. This type of time wasting problem is common for all the shared memory architectures and as such cannot be resolved by adapting the program. It can only be tackled by introducing a distributed memory architecture which is outside the scope of this semester thesis.

4.5 Training trajectories

To investigate the performance of the previously defined models, we have to collect the training and test data. Five Hardware in The Loop (HIL) simulations, each lasting 2 min, are conducted using the Pixhawk device and Mavros in order to record .bag files suitable for testing. For the inputs we try to use random maneuvers so as to keep the training procedure as simple as possible while at the same time we want to capture as many of the different flying regimes as possible. Figures 4.3 to 4.7 show the inputs and expected outputs of each trajectory.

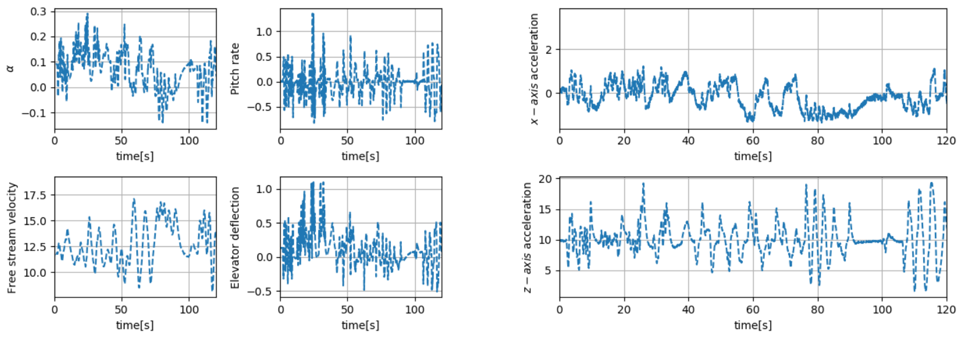

Figure 4.3: Inputs of the first trajectory on the left and outputs on the right

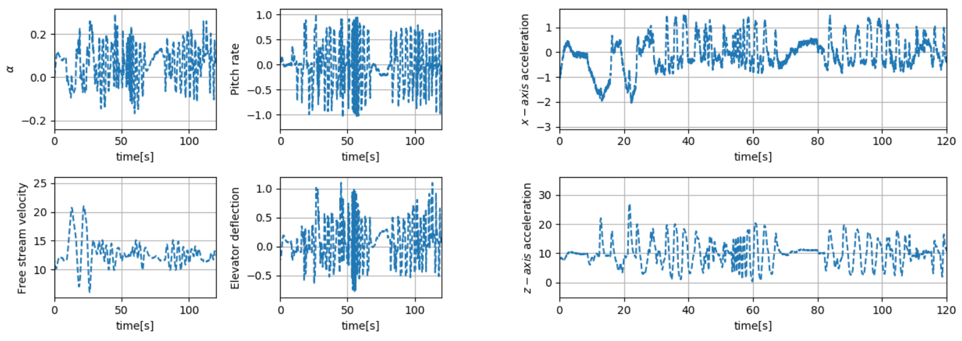

Figure 4.4: Inputs of the second trajectory on the left and outputs on the right

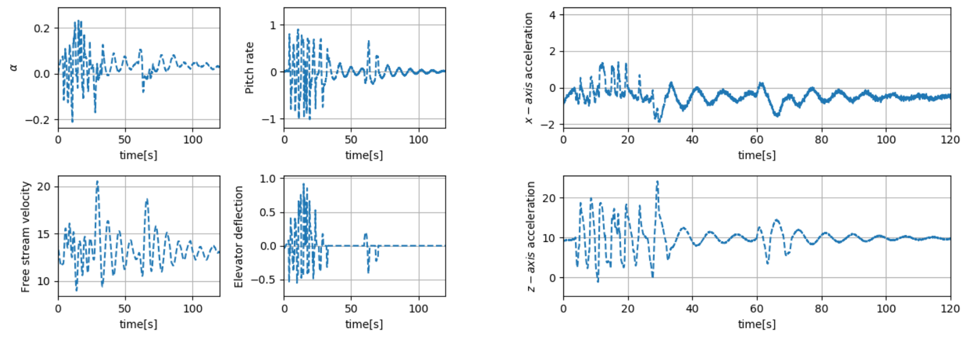

Figure 4.5: Inputs of the third trajectory on the left and outputs on the right

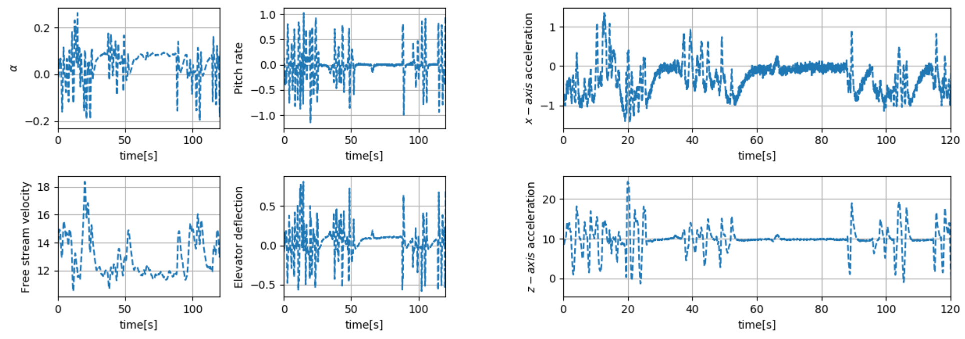

Figure 4.6: Inputs of the fourth trajectory on the left and outputs on the right

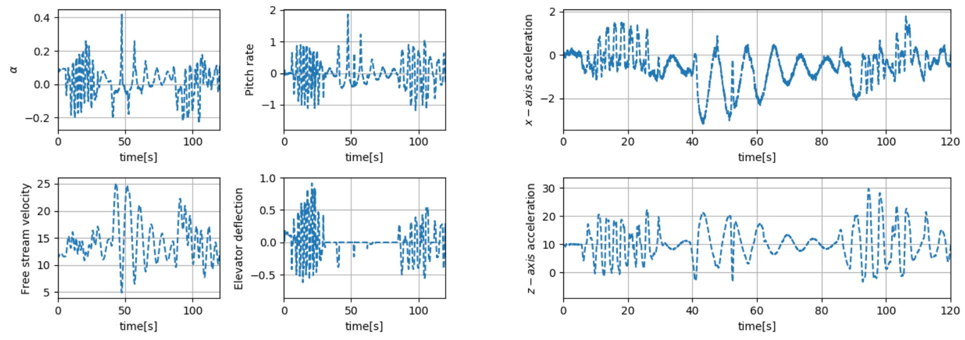

Figure 4.7: Inputs of the fifth trajectory on the left and outputs on the right

Chapter 5

Evaluation

Final chapter of the thesis presents the results obtained through simulations. Each of the three models is evaluated on five training trajectories and for all possible combinations of parameters such that:

- Learning rate satisfies: $\eta \in \{0.001, 0.005, 0.0075, 0.01\}$. Initially, a logarithmicaly spaced set of η has been introduced so as to give some intuition on what order of magnitude of η is reasonable for these setups. As a result, the interval between 10^{-3} and 10^{-2} has been chosen.
- Number of nodes in the hidden layers is in the set $\{10, 20, 40\}$. This set was chosen so as to limit the model's complexity and consequently the computational cost of the back propagation procedure.
- Window size N_W satisfies: $N_W \in \{40, 100, 150, 200\}$. This is equivalent to using last 2, 2.5, 7.5 or 10 seconds of measurements for the current training set. This is a reasonable set of training set lengths given the 2 min training procedure.

In total, 720 models have been trained for all the different sets of hyperparameters and training trajectories. These results are then utilized in order to perform the Grid Search over the proposed parameter space so as to find the model structure that generalizes the best. Design choices are compared by calculating the average root mean squared error (RMSE) of estimated a_x, a_z and q, where q is obtained by forward integration of the previously estimated derivative $\dot{q}$ and the initial condition is set to be equal to the previous measured value. Averaging of the RMSE is done over all the training trajectories and the used RMSE for a particular training trajectory is given by:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(y_k - \hat{y_k})^2} \tag{5.1}$$

where y can be any of the three estimated values and N is the total number of points in a training trajectory. Furthermore, let us define a combined RMSE for a system with multiple outputs as a sum of individual output RMSE. We will use the combined RMSE as a metric to compare the models when performing the Grid Search.

5.1 Grid Search

In this section we list the architectures that achieved the lowest average root mean squared error on the training sets for either of the three estimation outputs or achieve the lowest combined RMSE. The models are jointly given in the table 5.1.

Table 5.1: Best models

Min error for	Model	η	Nodes	N_W
a_x	Model 2	0.001	10	200
a_z	Model 3	0.005	40	200
q	Model 2	0.005	10	100
Combined	Model 2	0.001	40	200

The achieved minimal errors are jointly given in table 5.2.

Table 5.2: Achieved losses for the best models

Min error for	RMSE a_x	RMSE a_z	RMSE q	Combined
a_x	**0.3739**	1.8534	0.0469	2.2744
a_z	0.4244	**1.5209**	0.0464	1.9917
q	0.4707	2.6328	**0.0449**	3.1485
Combined	0.3845	1.5220	0.0455	**1.952**

We can now identify that for the proposed feasible set of parameters, gathered training trajectories and combined RMSE as the criterion function, the model that best generalizes on the provided data is a two branch Neural Network with two hidden layers in the first branch, 40 neurons in each hidden layer, that utilizes a window size $N_W = 200$ to prepare data for online training and uses $\eta = 0.001$ as an initial learning rate parameter. Interesting to note is the fact that the network that achieves the lowest combined RMSE does not coincide with any of the other networks. This shows that the obtained Grid Search result is highly dependant on the metric used for comparison.

5.2 Training loss convergence

In this section we address the question of estimating the average time of convergence for the optimal model. in Figure 5.1 we present the actual real time predictions of the optimal model during the training procedure. As this is just for the purpose of display, we chose to present the results on the model trained with fifth trajectory. The red dots show the estimated values at each update time of the model while in blue we give the measured values obtained from the IMU.

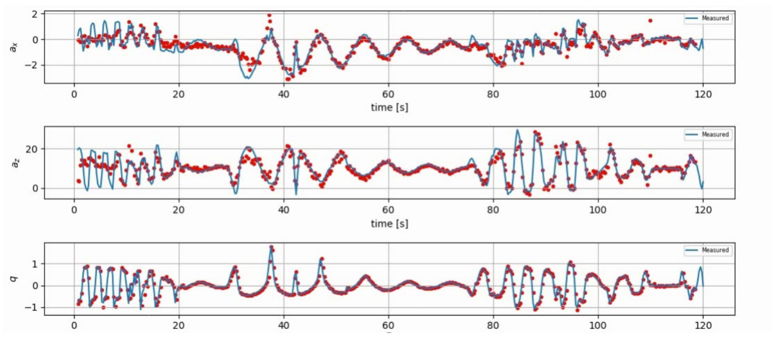

Figure 5.1: System predictions during training

After the initial period where the predictions significantly differ from the measured values, we can see that the system is working quite well at the later stages of the training procedure. In fact, we can look at the Figures 5.2 and 5.3 where the training loss curve is given for each of the 4 best models presented in table 5.1.

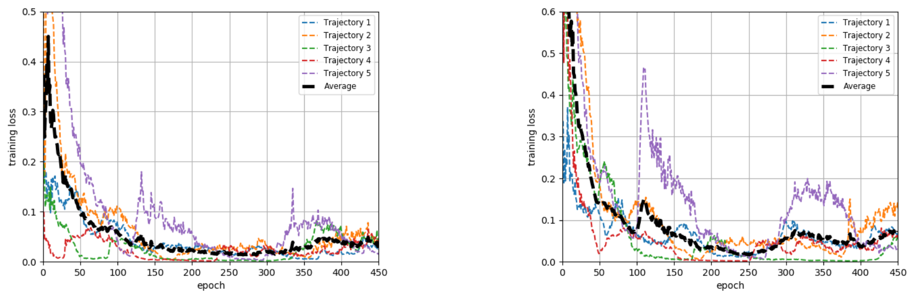

Figure 5.2: Best model for combined error on the left, best model for a_x on the right

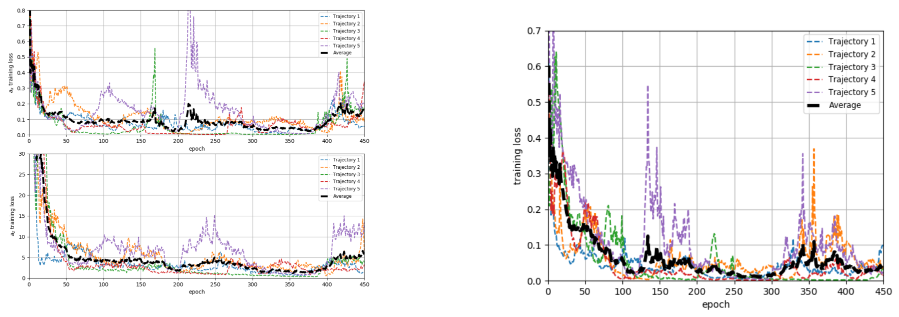

Figure 5.3: Best model for a_z on the left, best model for q on the right

Here, training curves for different training sets are colored differently. In each of the plots an extra black line is introduced that represents the average trend in the loss function. Note that the best model for a_z estimation has two loss functions depicted as this model has all the outputs separated into branches, As can be seen from the pictures, for all four observed models, the average training curve converges to a steady level. Using this curve we can estimate the average convergence time for each of the four models. Taking into account that each update period lasts 0.25 sec, the estimated convergence times are:

- Best a_x estimate: 43.75 sec
- Best a_z estimate: 25 sec
- Best q estimate: 50 sec
- Best combined estimate: 37.5 sec

5.3 Single parameter variation

In this section we will analyze what happens with the performance of the chosen optimal model if we vary each of the parameters individually. Table 5.3 shows the average RMSE if we keep the optimal parameters but change the value of learning rate.

Table 5.3: Varying learning rate

Learning rate	a_x RMSE	a_z RMSE	q RMSE
$\eta = 0.001$	**0.384**	**1.522**	**0.046**
$\eta = 0.005$	0.425	2.306	0.046
$\eta = 0.075$	0.450	2.851	0.047
$\eta = 0.01$	0.532	3.229	0.048

Since the average error increases for all the outputs, we can conclude that for this architecture increasing learning rate so as to potentially decrease the time needed for the model to converge can only degrade the overall performance of the system. Even though larger learning rates cause the training loss curve to decrease faster, this also makes the system react faster to the new data making it more sensitive to noise and potential outliers. In addition, we have to be careful as large learning rates can lead to oscillations during optimization procedure. The effect of increasing the learning rate is shown in Figure 5.4.

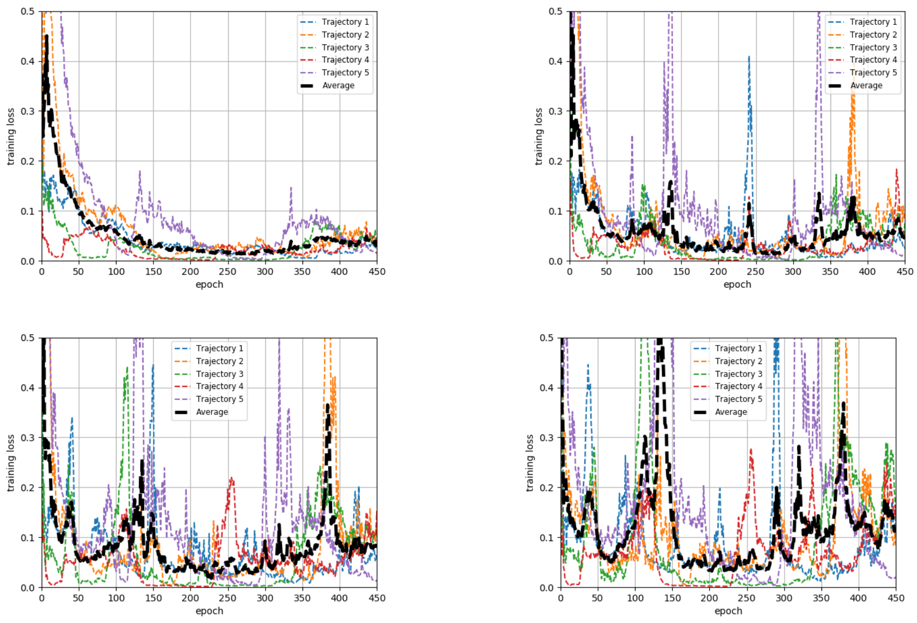

Figure 5.4: Training loss curves for $\eta = 0.001$, $\eta = 0.005$, $\eta = 0.0075$ and $\eta = 0.01$

The table also suggests that we could try even smaller values of η. However, trying learning rates of a lower order of magnitude then proposed results in slower convergence and gives no significant performance improvement. This can be observed in Figure 5.4, in the plots for $\eta = 0.001$ and $\eta = 0.005$ (for the other two plots it makes no sense to analyze convergence as the system is too sensitive to changes in the input signals). For $\eta = 0.005$ it takes the training loss curve approximately 12.5 sec to converge while for $\eta = 0.001$ it takes 37.5 sec.

Similarly, table 5.4 shows the error values if we change the batch size used.

Table 5.4: Varying batch size

Batch size	a_x RMSE	a_z RMSE	q RMSE
$N_W = 40$	0.654	3.145	0.054
$N_W = 100$	0.428	2.147	0.051
$N_W = 150$	0.440	1.956	0.047
$N_W = 200$	**0.384**	**1.522**	**0.046**

In this case, only RMSE of a_x does not monotonically decrease with the increase in batch size. As already described, larger batch sizes repeatedly use some sample points for multiple training iterations. This has helped our model in this setup but in general can also lead to overfitting to the training data. One possible solution that would allow larger batch sizes would be to introduce a forgetting factor that reduces the contribution of the older points to the averaged gradient. However, there is not much that can be done to decrease the computation time required for back propagation when the number of parameters is increased. The increase in average training epoch duration for larger batch sizes is presented in Figure 5.5 and shows how using the larger batch sizes can violate the real time requirements of the procedure.

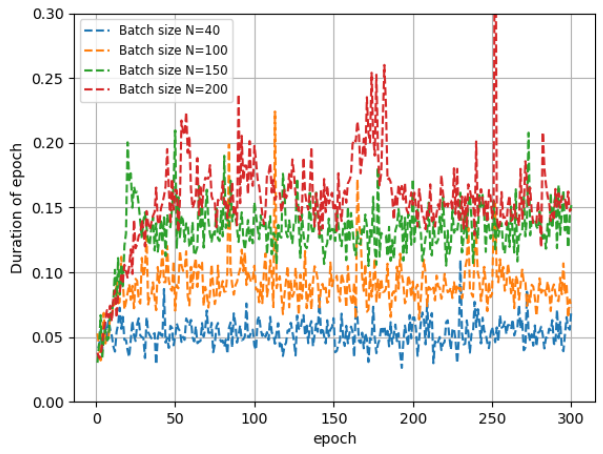

Figure 5.5: Average training epoch duration

Finally, we can vary the number of nodes in the hidden layers. The results are presented in Table 5.5

Table 5.5: Varying number of nodes

Number od nodes	a_x RMSE	a_z RMSE	q RMSE
10	**0.374**	1.853	0.047
20	0.375	1.664	0.046
40	0.384	**1.522**	**0.045**

If we take a closer look, we see that for a larger number of nodes the error for a_x increases whereas the error for a_z and q decreases. This shows that it was reasonable to test the Model 3 structure as well, where we could separately tune the number of neurons in each of the hidden layers. Even though we re-scaled the labels for the a_z output, in this case the optimization procedure still favours smaller average errors in the a_z estimation.

5.4 Validation loss convergence

To analyze the performance of the optimal model we also plot the validation loss curves. These curves are given for one model but for different training trajectories and for each of the validation sets. They are given in Figures 5.6 to 5.10 and show the RMSE achieved at different stages of the optimal model training procedure for each of the validation sets and each of the training trajectories.

Once again, we plot the curves that correspond to different training trajectories in different colors and add an extra black curve that corresponds to the average validation loss curve.

We note that irrespective of the chosen training data sets, the average RMSE validation loss decreases as the time progresses for each of the validation sets. The estimated value of the convergence time for the optimal model given in the section 5.2 agrees with the RMSE plots shown for different validation sets. In all of the cases, the validation loss has reached the steady level by the end of 100 training epochs. However, the loss trend does tend to decrease a bit more as the time passes but compared to the initial model estimation error, this decrease is insignificant.

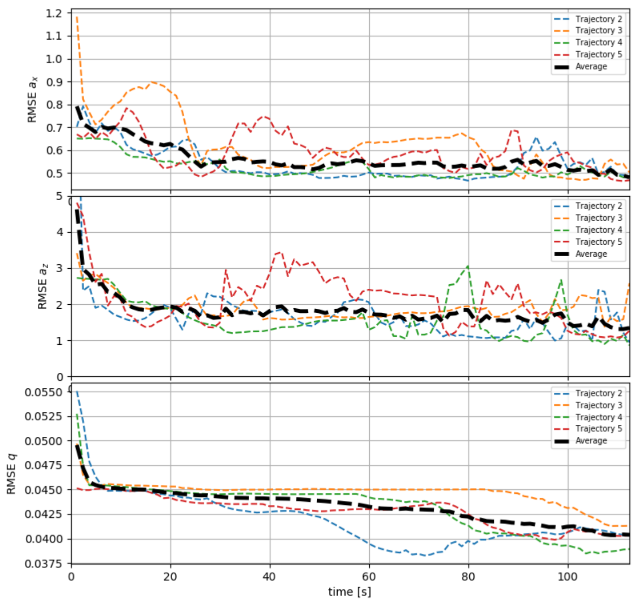

Figure 5.6: First trajectory used as a validation set

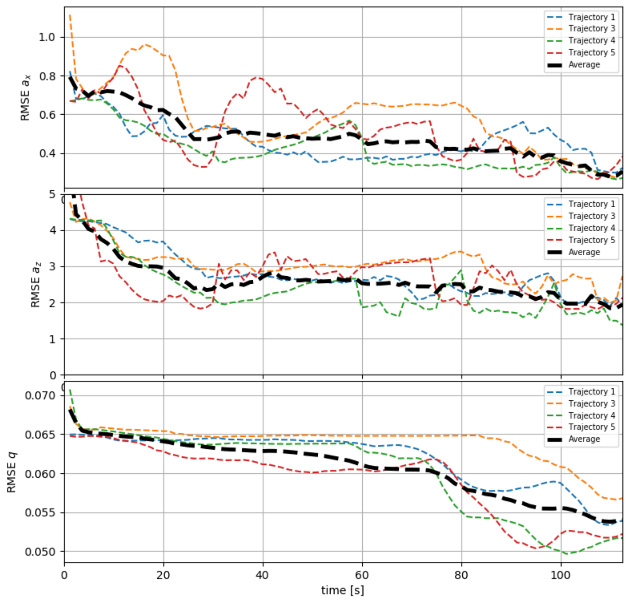

Figure 5.7: Second trajectory used as a validation set

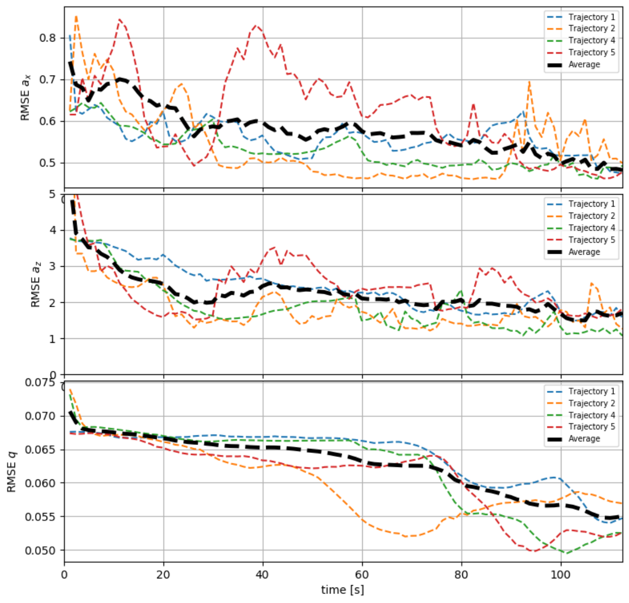

Figure 5.8: Third trajectory used as a validation set

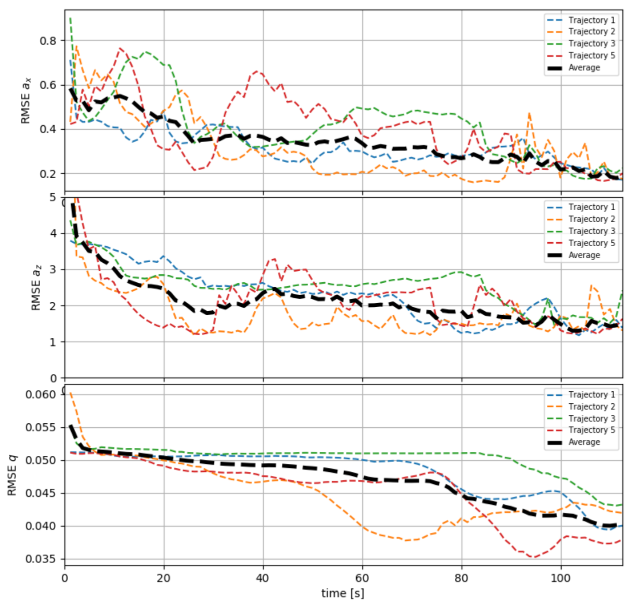

Figure 5.9: Fourth trajectory used as a validation set

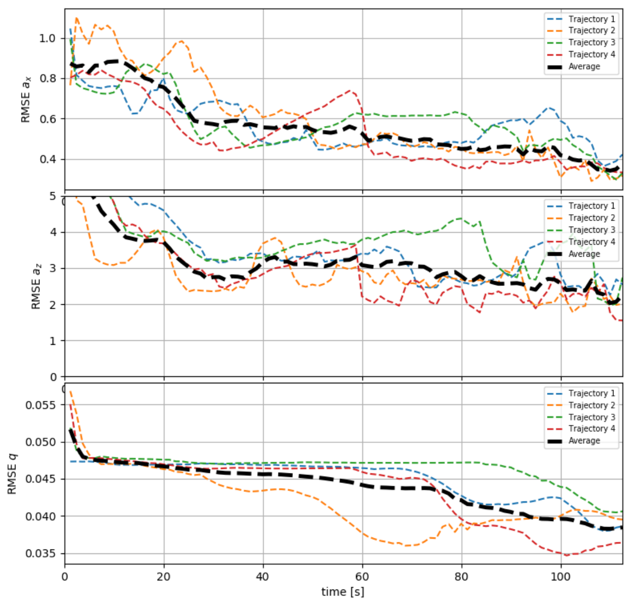

Figure 5.10: Fifth trajectory used as a validation set

5.5 Quality of the training trajectory

Validation loss curves can also be used to investigate the influence of training trajectories on the learning procedure. If the average validation loss curve starts to increase as the training progresses, one can consider this as one of the indicators that the input signal at that time interval is not good for the identification procedure. In all the HIL simulations we used random maneuvers in order to make the training procedure as simple as possible. Therefore, it is entirely possible that some of the regimes are not suitable for the training. In this section we show that care has to be taken while performing the maneuvers in order to prevent the fixed-wing UAV from violating the underlying aerodynamics model that we want to identify. By observing the Figures 5.6 to 5.10 we can get some intuition on what kind of design choices in the training maneuvers could cause a bottleneck for the training procedure. In particular, let us take a closer look at the a_x RMSE curves for the first, second and fourth validation set. We can see that for all three validation sets, models that are trained with the third and fifth trajectory experience some undesired behaviour. The observed sections are marked in red in Figure 5.11.

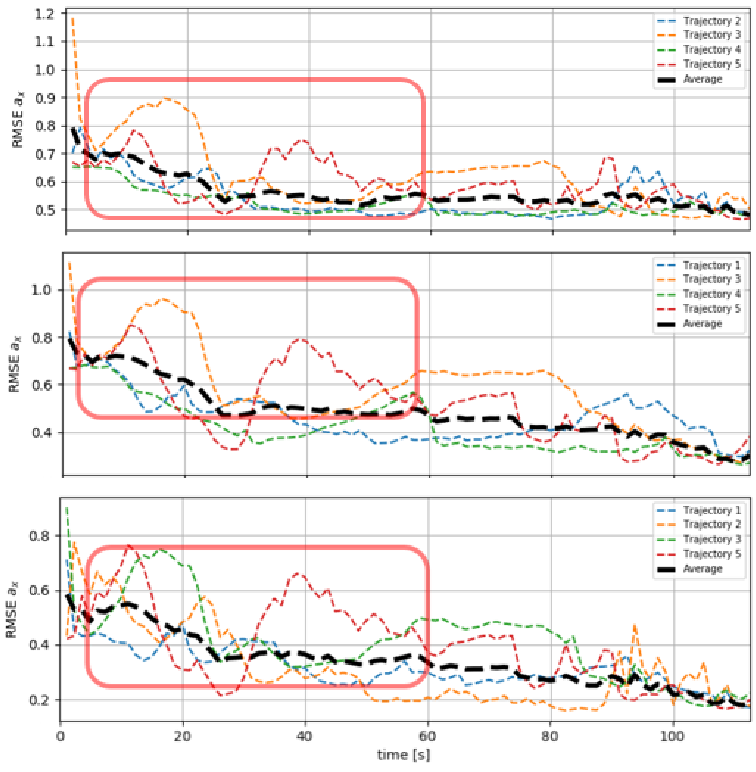

Figure 5.11: Sections with undesired behaviour

The models trained on the third trajectory experience problems in the first 80 epochs as there is an increase in the RMSE value for a_x. Similar problem experience the models trained on the fifth trajectory but in the range from 100 to 240 training epoch. These epochs correspond to approximately 0 to 25 sec time interval for the third trajectory and approximately 25 sec to 60 sec time interval for the fifth trajectory. In section 4.5 we defined all the trajectories that are being used. Figure 5.12 shows the inputs of the trajectory 3 and trajectory 5 and we mark in red the critical regions.

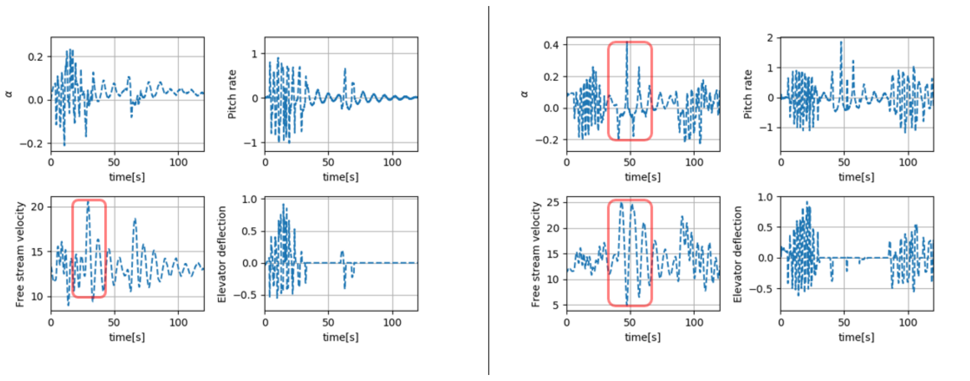

Figure 5.12: Critical regions in trajectories 3 and 5

We can now conduct a closer inspection of the signals in search for some specific elements that caused an undesired reaction from the system.

- For the third trajectory, we observe that in the proposed time period the free stream velocity signal significantly increases. The model has not seen this kind of input before and as a consequence is trying to learn a different dynamics model from it.

- Fifth trajectory experiences a similar problem as the third one. Around 50 sec a significant increase in the free stream velocity is observed. By looking at the equations (2.5) to (2.15), we infer that the longitudinal dynamics equations depend on the squared value of the free stream velocity. Therefore, it is no coincidence that the model is sensitive to changes in this input channel. Finally, if we take a closer look at the angle of attack we realize that the plane was actually stalling at the critical time interval which for sure gives a different aerodynamic model of the fixed-wing UAV.

One more thing to be discussed is overfitting, as it can occur due to badly designed training sets. We would like to investigate what would happen with the system if at a certain point in the training procedure we start receiving a constant input sequence for example. This is likely to happen as the level flight is for sure the most common type of aircraft movement. Such was the case with the fourth trajectory as presented in Figure 5.13. In the time span from 60 sec to 100 sec, which corresponds to the range from 240 to 400 training epoch, the aircraft is doing a level flight. If the system was overfitting we would see an increase in the RMSE for different validation sets. Fortunately, by looking at the figure 5.14, we can see that the level flight has no negative impact on the training procedure. This means that either our model design choices are good enough of a protection from level flight overfitting or that the time interval of the level flight required to trigger overfitting is considerably longer than 20 sec which is a priori not a very useful input design for system identification.

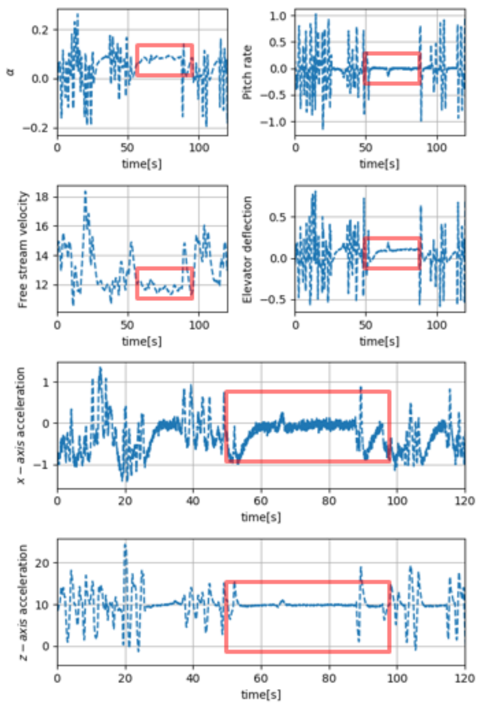

Figure 5.13: Critical section in trajectory 4

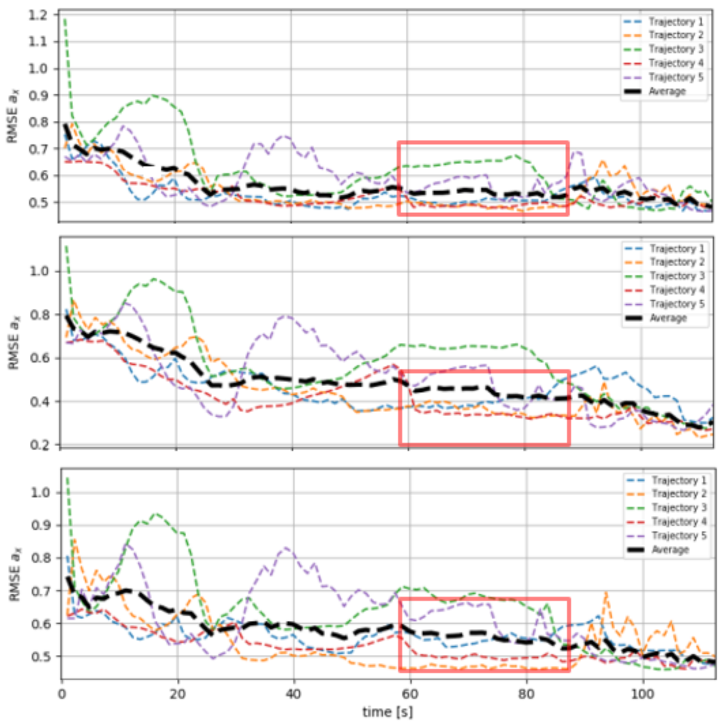

Figure 5.14: Critical sections in validation curves for trajectory 4

5.6 Estimation on the unseen validation data

Finally, we present the estimations provided by a fully trained model whose training procedure is shown in Figure 5.1. This corresponds to the optimal model identified with the Grid Search procedure, trained on the fifth trajectory. We use the remaining four unseen trajectories as the validation sets. Results are presented in Figures 5.15 to 5.18.

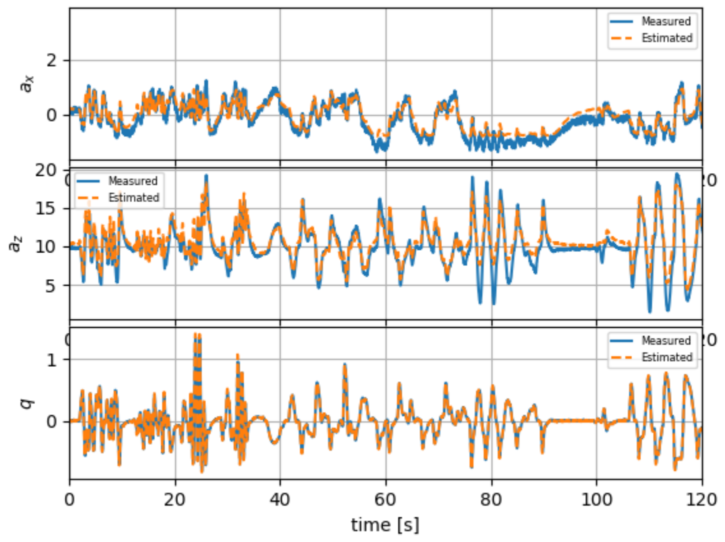

Figure 5.15: Fully trained model prediction on first trajectory

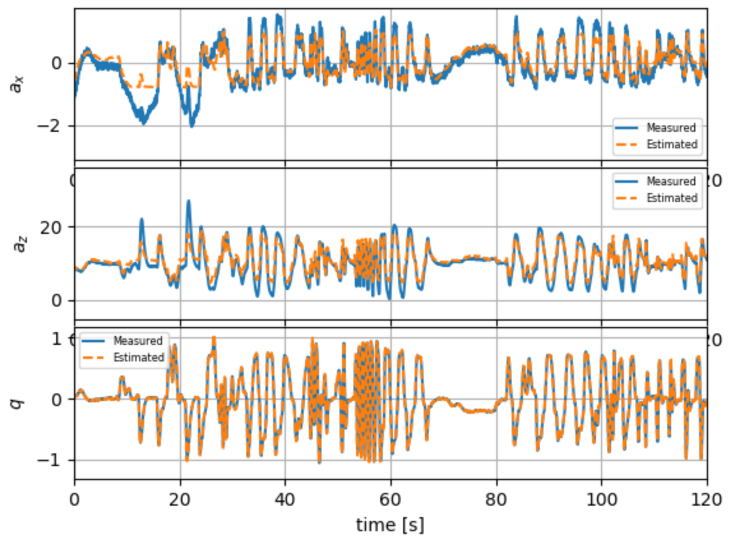

Figure 5.16: Fully trained model prediction on second trajectory

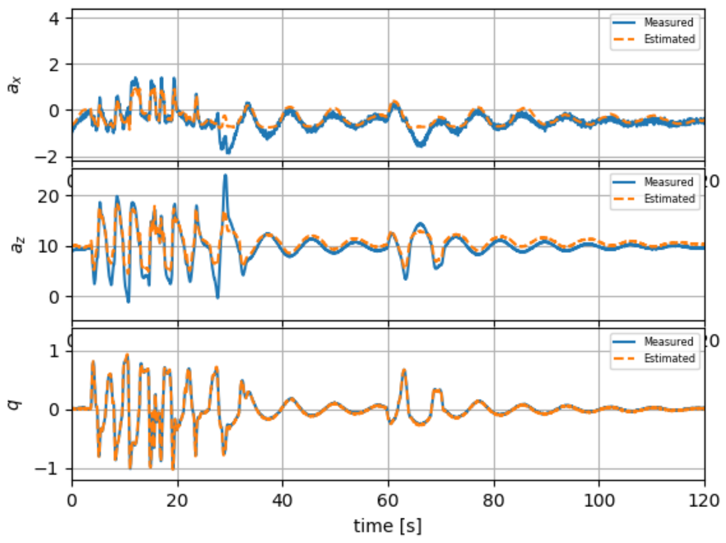

Figure 5.17: Fully trained model prediction on third trajectory

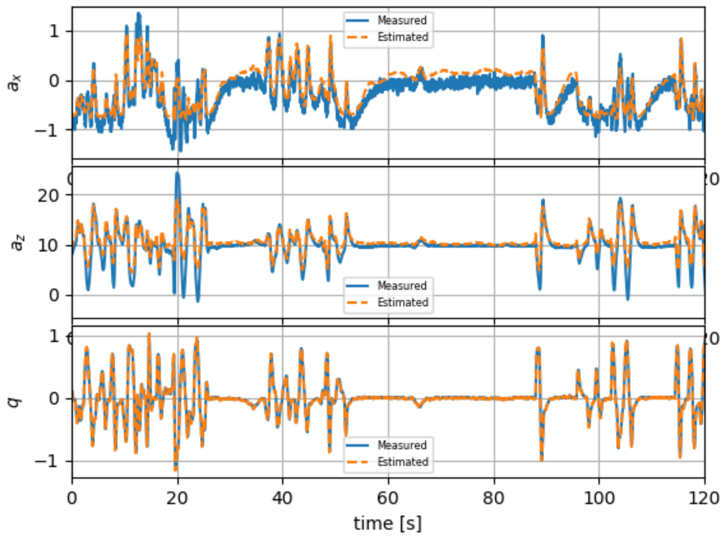

Figure 5.18: Fully trained model prediction on fourth trajectory

As we can observe, the measured and predicted values are a good match for the first, third and fourth trajectory. For the second validation set, the system is giving a good prediction of a_z but fails to predict some of the local extremes of the a_x. If we take a look at the inputs of the second trajectory given in Figure 4.4, we once again observe that the free stream velocity has higher values in the critical regions where the prediction is not accurate enough. Note that all of the validation sets have almost perfect pitch rate prediction. As mentioned before, the estimated value of q is obtained by forward integration of the previous $\dot{q}$ over the time interval between the previous and current measurement. Given that the measurements come from the IMU at a frequency of $f \approx 30$ Hz, the time interval between the two measured samples is very short for our estimate of $\dot{q}$ to play a significant part in the current estimate of q.

Chapter 6

Conclusion

In this thesis we proposed three different model structures capable of identifying the longitudinal dynamics model of a fixed wing UAV. An attempt was made to achieve balance between complexity of the model (in terms of number of tunable parameters), accuracy of prediction and required time of convergence. A feasible set of parameters has been defined for which a Grid Search algorithm has been applied in order to find the optimal set of hyperparameters for the given data sets. In total, 720 training simulations have been conducted. For the online simulation we had to previously gather the data in the form of .bag files. It was gathered through HIL simulation via randomly designed maneuvers with the aim to capture as many as possible flight regimes and to keep the training procedure as simple as possible. As a result, we had the opportunity to train and evaluate the models on different sets of data and to address some of the problems regarding the input design procedure. For the obtained optimal model structure we analyzed the influence of individual parameter variation, analyzed the convergence of the model for different training trajectories and gave an estimate of the average time required to converge. The optimal model which uses two hidden layers with 40 neurons to estimate the linear accelerations, learning rate $\eta = 0.001$ and window size $N_W = 200$ on average (estimated over the five training trajectories) takes 37.5 sec to converge. It has also been shown that a balance between the randomness of the training maneuver and the violation of the underlying dynamics model has to be found so that the training procedure is successful and no degradation in the quality of the model performance is observed. In this particular case, high values of the free stream velocities in the training sets tended to confuse the Neural Network and caused the model to worsen the estimation accuracy for all the validation sets.

6.1 Future Work

The proposed model gives satisfactory results in estimating the linear accelerations and the pitch rate derivative in terms of achieved average RMSE. This is important as it potentially opens the door to some further applications and improvements that require more time but are by all means achievable. Some of them include:

- A later dynamics model identification. Consequently, an attempt should be made to merge the two models into one that captures the entire dynamics of the fixed-wing UAV. In this scenario, a special care has to be taken as the real time requirements impose stricter limitations on the complexity of the model.

- Optimal input design should be investigated as well as it can potentially reduce the required convergence time of the system

- Finally, if an accurate complete dynamics model can be obtained, it can be used to estimate the throttle contribution to the total measured linear accelerations and we can use it to learn the model of the plane's engine online, without having to perform specialized tests.

Bibliography

[1] A. Jaimes, S. Kota, and J. Gomez, "An approach to surveillance an area using swarm of fixed wing and quad-rotor unmanned aerial vehicles uav(s)," 07 2008, pp. 1 – 6.

[2] M. Quigley, M. Goodrich, S. Griffiths, A. Eldredge, and R. Beard, "Target acquisition, localization, and surveillance using a fixed-wing mini-uav and gimbaled camera." vol. 3, 01 2005, pp. 2600–2605.

[3] J. Sun, B. Li, Y. Jiang, and C.-y. Wen, "A camera-based target detection and positioning uav system for search and rescue (sar) purposes," *Sensors*, vol. 16, p. 1778, 10 2016.

[4] S. Grogan, R. Pellerin, and M. Gamache, "The use of unmanned aerial vehicles and drones in search and rescue operations – a survey," 09 2018.

[5] P. Psirofonia, V. Samaritakis, P. Eliopoulos, and I. Potamitis, "Use of Unmanned Aerial Vehicles for Agricultural Applications with Emphasis on Crop Protection: Three Novel Case-studies," *Journal of Agricultural Science and Technology*, 2017.

[6] A. Dorobantu, A. Murch, B. Mettler, and G. J. Balas, "Frequency Domain System Identification for a Small, Low-Cost, Fixed-Wing UAV," 2011.

[7] R. Venkataraman and P. Seiler, "System Identification for a Small, Rudderless, Fixed-Wing Unmanned Aircraft."

[8] G. A. Garcia and S. Keshmiri, "Online artificial neural network model-based nonlinear model predictive controller for the meridian UAS," *International Journal of Robust and Nonlinear Control*, 2013.

[9] M. K. Samal, S. Anavatti, and M. Garratt, "Neural Network Based System Identification for Autonomous Flight of an Eagle Helicopter," 2008, proceedings of the 17th World Congress.

[10] N. K. Peyada and A. K. Ghosh, "Aircraft parameter estimation using a new filtering technique based upon a neural network and Gauss-Newton method," *The Aeronautical Journal*, vol. 113, 2009.

[11] V. R. Puttige, "Neural Network based adaptive control for Autonomous flight of fixed wing UAV," 2008.

[12] W. Zhang, "System Identification Based on Generalized Adaline Neural Network," *International Journal of Intelligent Control and Systems*, vol. 11, 2006.

[13] P. N. W. Lin, N. L. Kham, and H. M. Tun, "Longitudinal And Lateral Dynamic System Modeling Of A Fixed-Wing UAV ," *International Journal of Scientific Technology Research*, vol. 6, 4 2017.

[14] J. Qian, M. Nadri, P.-D. Morosan, and P. Dufour, "Closed loop optimal experiment design for on-line parameter estimation," June 2014.

[15] E. A. Morelli, "Flight test validation of optimal input design and comparison to conventional inputs."

[16] N. V. Hoffer, "System Identification of a small low-cost unmanned aerial vehicle using flight data from low-cost sensors."

[17] D. P. Kingma and J. L. Ba, "ADAM: A Method for stochastic optimization," 2015.

[18] *Robot Dynamics Lecture Notes*, 2018, robotic Systems Lab, ETH Zurich.