TU Clausthal
Clausthal University of Technology

DEPARTMENT OF INFORMATICS

Bachelor's Thesis

Modeling Quadcopter Dynamics using Recurrent Neural Networks

Author:	Yiheng CHEN
Matrikelnummer:	475374
Studiengang:	Informatik/Wirtschaftsinformatik (B.Sc.)
First Supervisor:	Prof. Dr. Sven Hartmann
Second Supervisor:	PD Dr.-Ing. habil. Umut Durak
Submission Date:	24 Juli, 2019

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig angefertigt und alle Quellen und Hilfsmittel vollständig angegeben habe.

Clausthal-Zellerfeld, 24 Juli, 2019

Signature:

Abstract

Recurent Neural Networks (RNNs) has shown its capability to process and predict time series data by storing long-term information. The data collected during the continuous flight of the quadcopter is also time series data, which can be processed by the RNN model. This thesis focuses on the implementation of RNN Long Short-Term Memory (LSTM) model, parameterization and result evaluation, which is used to process and predict quadcopter flight test data as a multivariate time series prediction problem. In this work, we used Crayzflie Nano Quadcopter as a tool for experimental flight data collecting, Keras and TensorFlow as the backend for LSTM model construction, and Mean Square Error (MSE) as metrics of model performance evaluation. In the experiment, we first flew the Crazyflie Nano quadcopter manuelly and collected the control input and acceleration data from Inertial Measurement Unit (IMU), then used the data to train our model and validated its prediction performance using test data different from the training data. By training the model and predicting the test data, the experimental results show that the LSTM model can accurately predict the acceleration of the next-time step, indicating that the LSTM model is an effective tool for quadcopter acceleration predicting and trajectory tracking. For future work, other RNN models, such as Gated Recurrent Units (GRUs), can be tested and their performance compared to the LSTM model to solve such problems.

Zusammenfassung

Rekursive Neuronale Netze (RNNs) haben gezeigt, dass sie Zeitreihendaten verarbeiten und vorhersagen können, indem sie Langzeitinformationen speichern. Die während des kontinuierlichen Flugs des Quadcopters gesammelten Daten sind auch Zeitreihendaten, die vom RNN-Modell verarbeitet werden können. Diese Arbeit konzentriert sich auf das RNN Long Short-Term Memory (LSTM), die Parametrisierung und die Ergebnisauswertung, bei der es sich um ein multivariates Problem der Zeitreihenvorhersage handelt. In dieser Arbeit verwendeten wir Crayzflie Nano Quadcopter als Werkzeug für die experimentelle Flugdatenerfassung, Keras und TensorFlow als Backend für die LSTM-Modellkonstruktion und Mean Square Error (MSE) als Metriken für die Bewertung der Modellleistung. In dem Experiment flogen wir zuerst den Crazyflie Nano Quadcopter manuell und sammelten die Daten von Inertial Measurement Unit (IMU), dann verwendeten wir die Daten, um unser Modell zu trainieren und seine Leistung zu validieren. Experimente zeigen, dass das LSTM-Modell ein effektives Werkzeug für die Vorhersage von Quadrocopter-Beschleunigung und Verfolgungstrajektorienverfolgung ist. Für zukünftige Arbeiten können andere RNN-Modelle, wie z. B. Gated Recurrent Units (GRUs), getestet und ihre Leistung mit dem LSTM-Modell verglichen werden, um solche Probleme zu lösen.

Contents

List of Figures

1 Introduction

Machine learning, especially Artificial Neural Networks (ANNs), has recently gained boost and prosperity. The fundamental unit of ANN is based on the first mathematical model of biological neurons called *McCulloch-Pitts Neuron* [3], which was introduced in 1943 by Warren McCulloch and Walter Pitts. Since then, the efforts of several generations of scientists, the mathematical model of artificial neural networks have continued to develop, and the hardware devices that support the operation of neural networks have also been greatly improved. Today, ANN is widely used in many fields, such as automatic speech recognition using Recurrent Neural Networks (RNNs) and image recognition technology using Convolutional Neural Networks (CNNs).

Quadcopter is multicopter powered by four rotors placed in the same plane. With the growth and development of Unmanned Aerial Vehicle (UAV), ANNs has also been used for dynamic control of quadrotors as a data-based robot controller solution complemented by expert designers of its nonlinear data processing capabilities. In [1], a simple feedforward neural network shows its performance in the quadcopter's flight trajectories control, which is not included in the training data. In [2], the researchers used three-minute training data from four motors to achieve six-second hover without a low-level controller by using reinforcement learning. These results show that ANN is a powerful tool for automating quadcopters. According to [15], Mohajerin have stated: "RNNs as a special class of ANNs possess the universal approximation property and in theory can reconstruct state-space trajectories of dynamic systems arbitrary well".

Inspired by data-driven, learning-based control schemes modelling introduced in [2], modeling using RNN for multivariate time series prediction of quadcopter is proposed in this work. By knowing of previous inputs data {thrust, pitch, roll, yaw} and angular and linear acceleration in time series, the acceleration in next time step can be predicted, which can furthermore be used to predict its space location during flight.

1.1 Goal of this work

Our goal is to solve nonlinear relationship between control inputs and acceleration of quadcopters by implementing six LSTM-based recurrent neural network models and solve it as time series forecasting problems. Building the RNN Long Short-Term Memory (LSTM) model separately for each acceleration as output instead of building a fully integrated model with all six accelerations as output is because, first, only a portion of input control has an effect on a single acceleration, which will be illustrated in the second section. Second, due to the communication bandwidth limitations between the PC client and quadcopter, the four control inputs and the six acceleration measurements cannot be collected simultaneously.

1.2 Methodology

With the development of open source machine learning, there are many frameworks can be used to provide advanced ANN modelling infrastructure for personalized solutions today. The most common used low-level frameworks are *TensorFlow*[1], *Theano*[2], *MXNet*[3], *Torch & Pytorch*[4], and high-level libraries are *Caffe*[5], *Keras*[6]. While low-level frameworks provide fine-grained but also complex mathematical operation, such as matrix multiplication, to implement neural network construction, high-level libraries use those low-level mathematical frameworks as backend libraries and provide more abstract modelling unit such as Layers for intuitiveness and succinctness of code. In the framework mentioned above, except for *Caffe*, the rest can support RNNs implementation.

The thematic introduction took place via a Machine-Learning-Mastery course on time series prediction[7] with *TensorFlow* and *Keras*. In this course, the topic was introduced with the programming language *Python*[8], with which the functioning of recurrent neural network could be described in a impressive way. Due to the good documentation and the provided source code, also to the efficient models, *TensorFlow* and *Keras* are selected for the practical implementation of the concepts of deep learning in this work.

1.3 Construction of the work

This work is divided into 7 chapters. Chapter 2 first gives an insight of the basis of quadcopter dynamic, describes the dependencies between state,

[1]https://www.tensorflow.org/, Accessed: Mai 2019
[2]http://deeplearning.net/software/theano/, Accessed: Mai 2019
[3]https://mxnet.apache.org/, Accessed: Mai 2019
[4]https://pytorch.org/, Accessed: Mai 2019
[5]https://caffe.berkeleyvision.org/, Accessed: Mai 2019
[6]https://keras.io/, Accessed: Mai 2019
[7]https://machinelearningmastery.com/, Accessed: Mai 2019
[8]https://www.python.org/, Accessed: Mai 2019

velocities and accelerations of quadcopter, and introduce six control-input acceleration models as target model for neural network construction in this work.

In chapter 3, classical learning method including supervised learning for ANN, time series prediction model for sequence data and the transformation of the format by these two learning methods are described.

Chapter 4 describes the solution for time series prediction problem with deep learning in this work. It begins with normal artificial neural networks illustration, then introduces a special type called RNN, and finally introduces a special class of RNN called LSTM.

Chapter 5 deals with the description of the concrete procedure and details of aircraft configuration, data processing and neural network architecture used in this work for model construction.

In chapter 6, we show the prediction result of all 6 neural network models for six linear and angular accelerations prediction respectively. We evaluate the result and analyse the advantages and disadvantages of these models.

Chapter 7 concludes the work with a conclusion.

2 Basis for Quadcopter Dynamic System

Understanding of quadcopter dynamic model contributes to further research in this work. In the fist subsection, we introduce two basic reference frames for quadcopter and its dynamic model parameters. In the second subsection, the interactions between control inputs and state of quadcopters are shown by modeling equations and a flow graph. Based on these two subsections, in the third subsection, we introduce our six target predictive models of neural network modeling for this work.

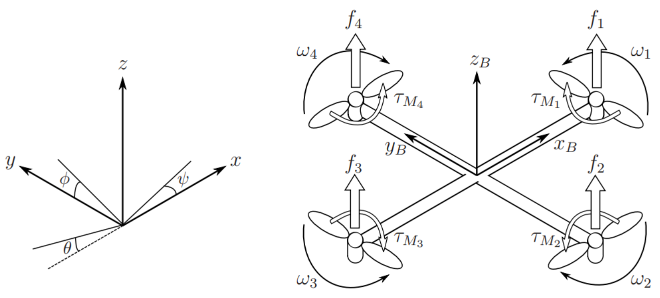


Figure 1: The inertial frame I (in left) and body frame B (in right) of a quadcopter. Taken from [4].

2.1 Reference Frame Model of Quadcopter

There are two reference frames for quadcopter attitude description, shown in Figure 1. In the North-East-Up inertial frame I, the position of quadcopter is presented as $\xi = (x, y, z)$ in the direction of corresponding axis of reference respectively. The linear velocities $\dot{\xi} = (\dot{x}, \dot{y}, \dot{z})$ and linear accelerations $\ddot{\xi} = (\ddot{x}, \ddot{y}, \ddot{z})$ are the first and second derivative of $\xi = (x, y, z)$ for time t, respectively. The attitude of quadcopter is presented by three Euler angles $\eta = (\phi, \theta, \psi)$, in which ϕ shows roll angles, θ determines pitch angle and ψ measures yaw angle between the x,y,z-axis in inertial frame I and body-fix frame B respectively. Similarly, the angular velocities and accelerations are denoted as $\dot{\eta} = (\dot{\phi}, \dot{\theta}, \dot{\psi})$ and $\ddot{\eta} = (\ddot{\phi}, \ddot{\theta}, \ddot{\psi})$, which can be calculated as first and second derivative of its attitude respectively. Four rotors, which are placed upwards in the same plane, generate thrust force $f_i (i = 1, 2, 3, 4)$ by its own angular velocity w_i. In this flight system, the torque created by rotors rotation movement also affect the attitude of quadcopter. Taken from [4], the thrust force f_i and torque τ_{M_i} are presented as:

$$f_i = k\omega_i^2, \tau_{M_i} = b\omega_i^2 + I_M \dot{\omega}_i \tag{1}$$

In which, k, b are constant and $\dot{\omega}_i$ is so small that can not be taken into consideration. I_M represent inertia moment.

2.2 Dependencies Between Flight State and Control Input

In this subsection, we illustrate the interaction between control inputs, linear and angular velocities and accelerations. Four dynamic equations are introduced firstly, then a graph is shown followed by equations for illustration of the relationships between those parameters. These relationships provide the argument for the correctness and reasonableness of models for this work in the third subsection.

Four rotors with its angular velocity ω_i generate total thrust force T, which impacts the acceleration and movement of quadcopter along z-axis in B. Taken from [4], thrust force T and thrust in body frame B are expressed mathematically as :

$$T = \sum_{i=1}^{4} f_i = k \sum_{i=1}^{4} \omega_i^2, T^B = \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} \tag{2}$$

The torques of four motors generate a whole torque τ_B on the quadcopter, which can be divided into three part torques on the direction of three axis in reference frame. By adjusting the angular velocities of diagonal rotors,the attitude movement rolling, pitching and yawing can be achieved. In the equation, the length between the axis of rotors and the enters of gravity of the entire aircraft is denoted as l. Taken from [4], τ_B is calculated as:

$$\tau_B = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} lk(-\omega_2^2 + \omega_4^2) \\ lk(-\omega_1^2 + \omega_3^2) \\ \sum_{i=1}^{4} \tau_{Mi} \end{bmatrix} \tag{3}$$

Taken from equation 20, page 6 in [4], the equation can be obtain by derivation of angular Euler-Lagrange equations which is about relationship between torques and Coriolis term.

$$\ddot{\eta} = J^{-1}(\tau_B - C(\eta, \dot{\eta})\dot{\eta}) \tag{4}$$

Taken from equation 21, page 6 in [4], the following equation describe aerodynamical effects which involve air resistance for realistical environment simulation. In the equation, $S_x = sin(x)$ and $C_x = cos(x)$, A_x is defined in inertial frame for representation of drag force coefficients.

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = -g \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \frac{T}{m} \begin{bmatrix} {}_{\psi}S_{\theta}C_{\phi} + S_{\psi}S_{\phi} \\ S_{\psi}S_{\theta}C_{\phi} -_{\psi} S_{\phi} \\ C_{\theta}C_{\phi} \end{bmatrix} - \frac{1}{m} \begin{bmatrix} A_x & 0 & 0 \\ 0 & A_y & 0 \\ 0 & 0 & A_z \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \tag{5}$$

According to the defined interactions in equations above, we can summarize the relationships between control inputs, angular velocity and acceleration, linear velocity and acceleration shown in Figure 2 (See [4] for detailed deduction). Linear acceleration $\ddot{\xi}$ drives linear velocities $\dot{\xi}$ by $\dot{\xi}^{t+1} = \dot{\xi}^{t} + \ddot{\xi}^{t} \triangle t$, and reverse linear acceleration $\ddot{\xi}$ depends on linear velocities $\dot{\xi}$ and control input ω. The same for angular velocities. Four motors generate thrust, which affect linear accelerations directly and torque, which determine the change of attitude and cause linear accelerations indirectly.

$$\begin{tikzcd}
 & T \arrow[rr] & & \ddot{\xi} \arrow[r] & \dot{\xi} \arrow[l, bend right] \arrow[r] & \xi \\
\omega \arrow[ur] \arrow[dr] & & & & & \\
 & \tau \arrow[r] & \ddot{\eta} \arrow[r] & \dot{\eta} \arrow[r] \arrow[l, bend left] & \eta \arrow[uul] \arrow[ll, bend left] &
\end{tikzcd}$$

Figure 2: Dependences between flight state, state derivatives and control inputs ω. Taken from [4].

2.3 Input Control Sequence Modeling

Quadcopter is controlled by four basic inputs: thrust command, roll command, pitch command and yaw command, denoted respectively as u_1, u_2, u_3, u_4, in which u_1 represent the thrust movement in the direction of the z-axis in B, and u_2, u_3, u_4 for rotation around the x-axis, y-axis and z-axis in B respectively. For the controlling, u_1, u_3 have affect on $\ddot{x}$ together, in which the u_1 represents the sum force of four rotors in direction of z-axis in B and u_3 affects the rate of movement of angle of z-axis in B in regard to z-axis in I. Similarly, u_1 and u_2 impact together $\ddot{y}$ and u_1 have influence on $\ddot{z}$. For angular acceleration, we have u_2 affects body rolling rates $\dot{r}$, u_3 impacts body pitching rates $\dot{p}$ and u_4 impacts body yawing rates $\dot{q}$ in body fixed frame. It should be noted that, unlike the previous basis reference background, both linear and angular acceleration $(\ddot{\xi}, \ddot{\eta})$ is defined B, because we measure the acceleration based on the IMU sensor on quadcopter instead of measurement unit in inertial space. These six models are presented as:

$$u_1^t, u_3^t \xrightarrow{state: \ddot{x}^{t-(1\ldots n)}, u_1^{t-(1\ldots n)}, u_3^{t-(1\ldots n)}} \ddot{x}^t \tag{6}$$

$$u_1^t, u_2^t \xrightarrow{state: \ddot{y}^{t-(1\ldots n)}, u_1^{t-(1\ldots n)}, u_2^{t-(1\ldots n)}} \ddot{y}^t \tag{7}$$

$$u_1^t \xrightarrow{state: \ddot{z}^{t-(1\ldots n)}, u_1^{t-(1\ldots n)}} \ddot{z}^t \tag{8}$$

$$u_2^t \xrightarrow{state: \ddot{\phi}^{t-(1\ldots n)}, u_2^{t-(1\ldots n)}} \ddot{\phi}^t \tag{9}$$

$$u_3^t \xrightarrow{state: \ddot{\theta}^{t-(1\ldots n)}, u_3^{t-(1\ldots n)}} \ddot{\theta}^t \tag{10}$$

$$u_4^t \xrightarrow{state: \ddot{\psi}^{t-(1\ldots n)}, u_4^{t-(1\ldots n)}} \ddot{\psi}^t \tag{11}$$

We shall use these six models in this work for artificial neural networks construction. Because the states of velocities are derivable from the input in previous time steps, it is sufficient to input the acceleration and input control in previous time steps for state modeling.

3 Learning Methode

3.1 Supervised Learning

Supervised learning is a major technique in machine learning. By training a model with labeled datasets, which consist of inputs value X and correct correspondent output Y, prediction model will learn the pattern or mapping relationship of a particular problem, and be able to make prediction of output value with feeding inputs value. According to the outputs type, supervised learning could be divided into regression problems and classification problems. When outputs variable is discrete value or category, for example "orange" or "apple", it is then classifications problem. If the outputs variable is a continuous value, the problem belongs to regression problems. The forecasting of quadcopter's acceleration in this work is a regression problem, because the accelerations are continuous values. For each data point in this work, it consists of X, which includes current control input and previous flight information, and Y including current linear or angular acceleration. With supervised learning format, collected flight data can be then processed by ANN in this work.

3.2 Time Series Prediction Problems

A time series is a sequence of observations taken sequentially in time [6]. It means that the observations in a time series has dependence in terms of time. Developing a model to describe a time series and explain the logic cause behind the data plausibly is involved in time series analysis, which is prevalent in classical statistics. According to [7], the major components of the data in time series are

- Level
- Trend
- Seasonality
- Noise

*A*nd the data in time series can be described as:

$$y = level + Trend + Seasonality + Noise \tag{12}$$

Flight is a continuous process, so the data collected during a whole flight is also sequence data. Velocities, linear and angular accelerations, space location have dependency to each other and to itself in time scale. By learning previous data, we can then make the prediction for the future movement.

Multivariate time series is time series in which more than one variables are collected in one dataset simultaneously. In the forecasting problem, according to the numbers of inputs variables which are measured and used for model training, and output variables which should be predicted, the time series forecasting problem can be grouped into uni-variate or multivariate inputs and uni-variate or multivariate outputs problems. The data collected in this work belongs to multivariate time series, because it consist of multidimensional data of controlling and acceleration. Since we want to use previous control and acceleration data to predict only acceleration in the future, the problem we focus in this work is a multivariate inputs and uni-variate outputs prediction problems.

One- or Multi-Step time series forecast describe how many time step ahead does a model make the prediction. One-step forecast is to predict outputs variable of t+1 at the time of t. Similar to multi-step prediction, it means predicting output variables at several time steps before the current time. We focus on one-step forecast for its best performance shown in this work.

3.3 Transformation from Sequence Data to Supervised Learning Dataset Format

Since supervised learning is one of the most popular and practical learning method in machine learning, which can be used for neural network training, we need to transform the collected time series into labeled supervised dataset, which can be processed and used for training our model in this work.

The *Sliding Window Method* is used in this work for data transformation. At time t, we have the observation$\{thrust_t,\ pitch_t,\ accelerationX_t\}$ in raw time series data, and we use sliding windows width w, then the inputs variable X of transformed supervised learning data at time t is $\{thrust_{t-w},\ pitch_{t-w},\ accelerationX_{t-w}.....thrust_{t-1},\ pitch_{t-1},\ accelerationX_{t-1},\ thrust_t,\ pitch_t\}$ and output y is $\{accelerationX_t\}$. In this way, each data sample has features of input data and acceleration in previous w time steps and labeled with current acceleration.

4 Application of Deep Learning

ANNs can be an effective solution for time dependent problem mentioned in previous section. Two types of ANNs have benn used widely for flight dynamics system control and research, Feed-Forward Neural Networks (FNNs) and RNNs. According to [15], while FNNs constitute a rich class of static maps, RNNs implement a rich class of mapping because of feedback within its architecture. In this chapter, we introduce firstly RNNs and then its special advanced class LSTM architecture.

4.1 Recurrent Neural Network

RNNs is a kind of promising mathematical models for sequential data processing. In contrast to FNNs, RNNs model allow neurons in hidden layers transfer information to themselves for historical information gathering. By introducing a sequence of hidden state, which means by feeding a inputs value, the recurrent neural network will give a output and meanwhile transfer into another state. With the same inputs value, different states will generate different outputs. With this ability of understanding the context of the inputs, RNNs is widely used in the problem involving sequence data prediction, for example language translation, speech processing and stock market prediction. Figure 3 shows a simple recurrent neural network structure with only one hidden layer.

Forward Pass In this procedure, data pass through the entire neural network and hidden layer get input from external source and output of hidden layers from previous time step in single hidden layer network. In a model

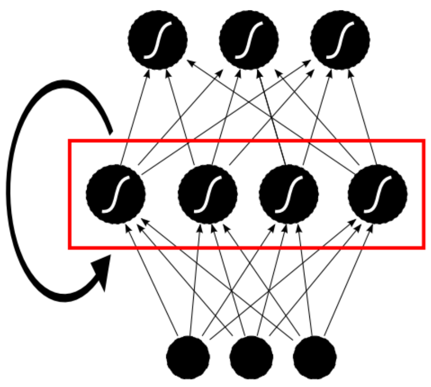


Figure 3: Recurrent neural network with single self connected hidden Layer, Taken from [8].

with I neurons in input layer and H neurons in hidden layer, x_i^t denotes the input value of neuron i at time t, w_i^h denotes the connection weight from neuron i to h, a_h^t denotes the value received by a neuron h at time t, $b_{h'}^{t-1}$ denotes the activation neuron h' in hidden layer at time t-1. Then we have the input value of neuron in hidden layer in following equation, which are taken from [8]:

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1} \tag{13}$$

Activation value of neuron h at time t can be then acquired by applying activation function θ_h of neuron h as

$$b_h^t = \theta_h(a_h^t) \tag{14}$$

Backward Pass One of the famous algorithm for RNN backpropagation calls Backpropagation Through Time (BPTT) [14]. In this procedure, neural unit update its weight and bias according to both the current feedback from output layer and from hidden layer one time step after. The mathematical equations for backward pass can be presented following, which is taken from [8]:

$$\delta_h^t = \theta'(a_h^t)(\sum_{k=1}^{K}\delta_k^t w_{hk} + \sum_{h'=1}^{H}\delta_{h'}^{t+1} w_{hh'}) \tag{15}$$

in which, δ_h^t denotes the gradient at time h.

Gradient Vanishing Problem in BPTT One of problem which cause poor performance of long term information dependence in classical RNNs is known as Gradient vanishing problem by using gradient descent algorithm. Due to upcoming of new information from more recent time, information from previous time will be overwritten from neural networks memory. Gradient vanishing problem is a common problem in Backpropagation algorithm. In FNNs, the gradient decrease drastically in direction from output layer to input layer, so that the shallower layers can learn subtly during training procedure. Similarly, in recurrent neural network, gradient decrease not only between layers, but also in alongside time scale because of cyclic connect of hidden layers. With gradient vanishing in time dimension, network can't learn from previous input in a long term. Figure 4 shows how the gradient vanishing problem occurs over a period of time.

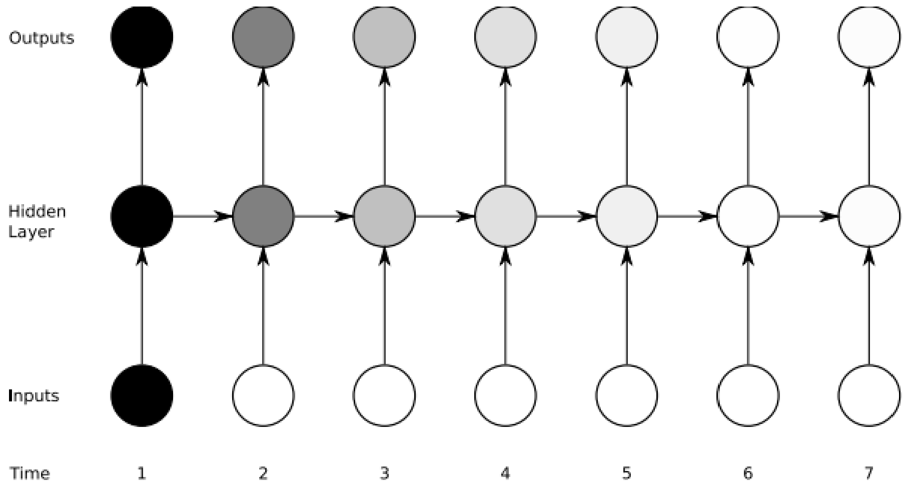


Figure 4: Gradient Vanishing Problem for RNNs. The grad of shadow shows the sensitivity to the input value of time 1. Because of new data coming , the neuron in hidden layers gradually loss the sensitivity for information of first input(from black to white). Taken from [8].

4.2 Long Short Term Memory Architecture

Long Short Term Memory networks (LSTM) is a powerful RNN architecture by learning long-range dependencies [12]. LSTM solves gradient vanishing/exploding problem and shows comparable the best performance with Gated Recurrent Unit (GRU) architecture among ten thousand architectures in practical use, according to [9]. LSTM was invented by German computer scientists Sepp Hochreiter and Juergen Schmidhuber in 1997 [13]. With the rise of deep learning since 2012, LSTM network have been successively improved by several scientist, such as Felix Gers and Fred Cummins for their contribution of Forget Gate [5]. Until now, systematic and complete LSTM frameworks have been developed and are widely used in many areas. LSTM architecture overcome the problem of memory overwritten by new information and able to keep old information for long time. Useful information can be kept cross many time steps, for example a pattern learned from timestamp 1, can be transfered at timestamp 20.

LSTM module is the basic repeating unit for information storage and processing in LSTM neural network. Figure 5 shows a LSTM module with information flow. Compared to normal RNN model with only one state, there are two states in LSTM model at time t: cell state c_t and short term state h_t. The former is for long term memory which carries the information from the beginning of training and the latter is much more relative to last time step. At each time t, there are three input values for the neuron: C_{t-1}, h_{t-1}, X_t and two output values: C_t, h_t.

For controlling the information flow, the gate based on two squashing function. The hyperbolic tangent squashing function (tanh) output values

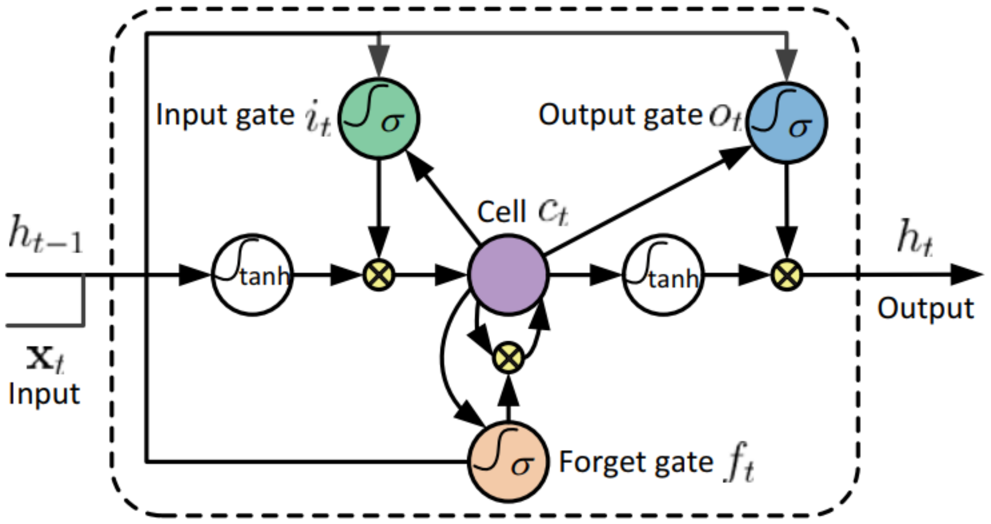


Figure 5: LSTM Unit. Taken from [11].

in -1 to 1 and sigmoid squashing function output values in range between 0 and 1. This two squashing functions are shown in Figure 6.

Because of the gate mechanism in LSTM module, LSTM neural network architecture is capable to insert or forget information to its long term memory for information flows regulation. The information flow controlled by two components: fully connected layers with sigmoid or tanh squashing function and vector point-wise calculation operation. The output of sigmoid layer are vector of number in range of zero to one indicating the percentage of data which are allowed to go through. Then the second component matrix calculation will operate pointwise multiplication of sigmoid layer outputs with the information which aim to pass through the gate. There are three types of gates in LSTM called: input gate, output gate and forget gate.

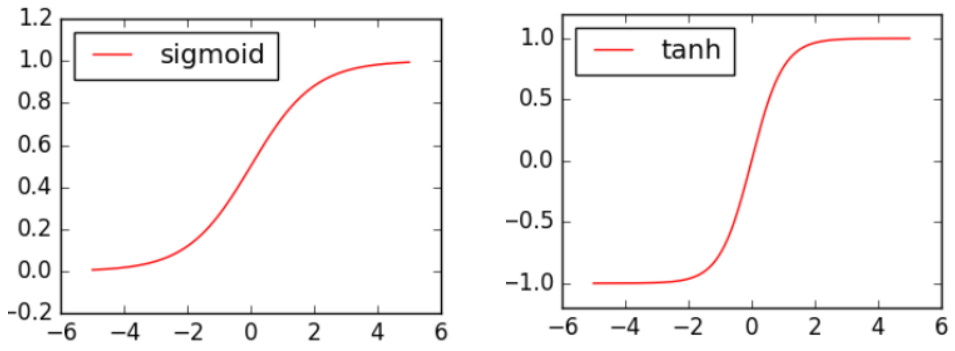


Figure 6: Sigmoid and Tanh Squashing Function, used in LSTM unit.

Forget Phase In the begin of information processing in LSTM unit at time t, part of non-significant information are dropout from previous cell state c_{t-1}. A neuron layer with sigmoid activation function(σ_1) take previous short term state h_{t-1} and current input values x_t as input, and return a vector f_t with numbers in range of 0 to 1, which indicating the forget index for former cell state. Vector f_t and c_{t-1} has the same size. The previous cell state c_{t-1} will be point-wise multiplied by f_t for part information deletion. Multiplied by 1, the cell inforamtion is totally kept and multiplied by 0, the information is totally forgotten.

Input Phase In this phase, new information are written to the current cell state for information update. Input gate and tahn layer are involed in this phase. Input gate is a full connected layer with sigmoid activation function(σ_2) which generate a vector i_t with numbers in range [0,1]. A full connected layer with tanh activation function($tahn_1$) generate a vector serving as candidate values for update. Vector i_t controls which value in candidate vector can be added into new cell state by pointwise multiplication with it.

The new cell state is then generated by deleting some information of previous cell state and adding some new information into them.

Output Phase After updating of cell state by forgetting and adding data, the cell state C_t is suitable for output prediction in current time step. Output gate is a full connected layer with sigmoid activation function(σ_3). By taking the previous short term state h_{t-1} and current input x_t, it outputs a vector o_t with numbers in range [0,1] for output activation. Regulated by a fully connected layer with tahn activation function ($tanh_2$), current cell state is multiplied by o_t for generation of output h_t. h_t is transmitted both to output layer as prediction and hidden layer in next time step t+1 as short term state.

The mathematical expressions of data processing in LSTM unit at time t can be computed with following equations, which are taken from [10]:

$$f_t = \sigma_1(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{16}$$

$$i_t = \sigma_2(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{17}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{18}$$

$$o_t = \sigma_3(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{19}$$

$$h_t = o_t \odot tanh_2(C_t) \tag{20}$$

in which, W denotes the weight matrix between two neural layers, and the bias term is denoted as b. Input gate, forget gate, candidate layer, output gate and output layer are denoted as i, f, c, o and h respectively. $\odot$ denotes point-wise product. Four weight matrixes and bias adjust the input x_t during learning phase for error minimization.

5 Experiments

5.1 Flight Tools Configuration

Crazyflie Nano Quadcopter *Bitcraze Crazyflie 2.0(CF)*[9] is a small size quadcopter developed by *Bitcraze* Company from Sweden. Both hardware architecture and software of CF are open-source. We choose this nano-quadcopter as our research instrument mainly for the following specifications:

- Weight only 27 grams
- Flight Data collection using PC client
- Three-axes IMU-gyroscope for angular acceleration measurement
- Three-axes IMU-accelerometer for linear acceleration measurement

Crazyflie Communication Beside CF, Crazyflie PC client and Crazyflie iOS Client are also used in this work for flight data collection. Crazyflie PC Client is used for data receive and Crazyflie iOS Client is for CF manuelly flight control. Crazyflie PC Client is written by python. With the help of PC client, flight data measured by Crazyflie onboard IMU sensors can be recorded and sent to PC computer. The communication between PC client and CF is through *Bitcraze Crazyradio PA USB dongle.* Crazyflie can be meanwhile connected and controlled by mobile phone application through bluetooth.

After successful connection between PC client and Crazyflie, we can find the available log variables in Log TOC tab and add the measurements we

[9]https://www.bitcraze.io/crazyflie-2/ Accessed: Mai 2019

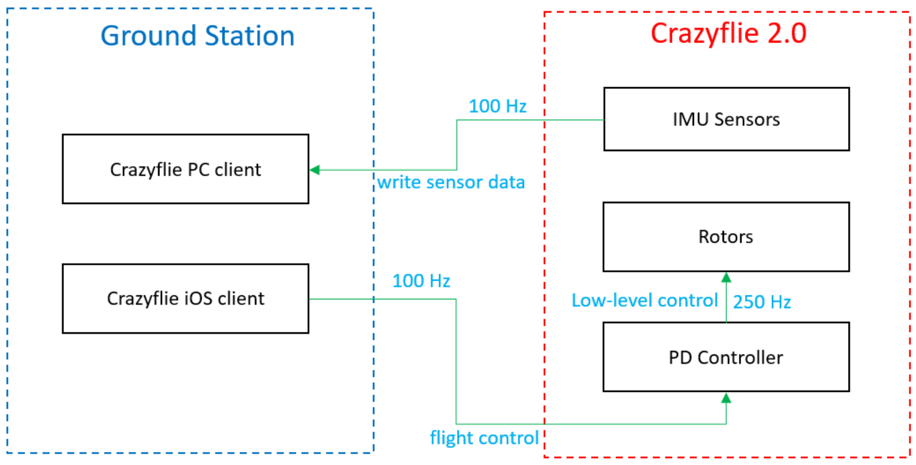


Figure 7: Connection between client and Crazyflie.

want into a logging block. We can find the self-defined block in Log Block tab and choose when to start and record the log blocks. Each measurement in the same block will be recorded simultaneously and with same timestamp. Each blocks can transfer data in the limitation of maximum of the bandwidth of the radio.

Crazyflie Firmware Flashing In the original source code of CF, the angular and linear acceleration are available, but the input values not. In order to access it, we need to modify its firmware source code. For doing that, the development environment must first be set up. In the following, we will discuss which steps are necessary for this procedure[10]:

1. start the virtual machine of Crazyflie

[10]https://forum.bitcraze.io/viewtopic.php?f=11&t=3269 Accessed: Mai 2019

2. load scouce code from GitHub

3. start the editor and open the firmware file

4. add the following source code in stabilizer.c file:

    (a) LOG_ADD(LOG_FLOAT, roll, &control.roll)

    (b) LOG_ADD(LOG_FLOAT, pitch, &control.pitch)

    (c) LOG_ADD(LOG_FLOAT, yaw, &control.yaw)

    (d) LOG_ADD(LOG_UINT16, thrust, &control.thrust)

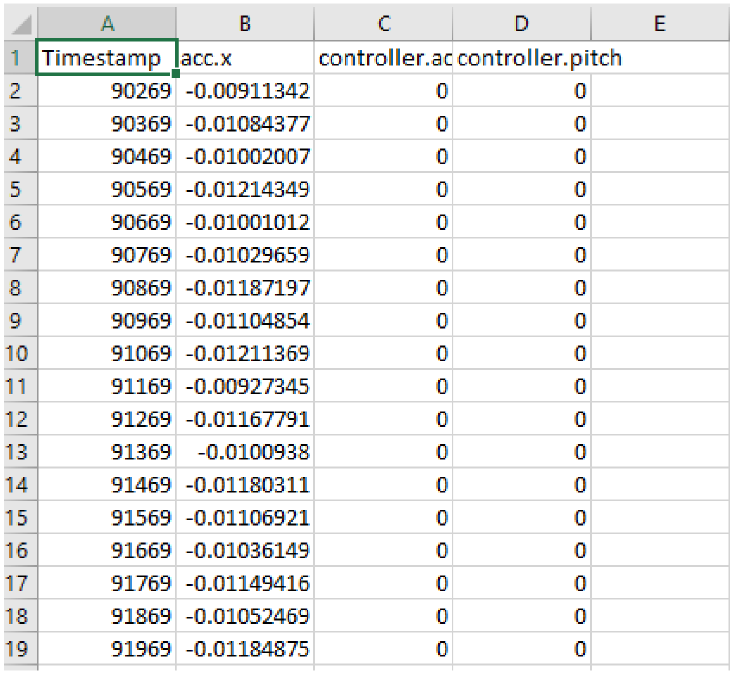
5. build the source code into binary files which can be processed in CF

6. flash the changed source code into CF via Crazyradio dongle

5.2 Data Preprocessing

Data Collection and Description For collecting flight data to train our RNN model, CF was controlled to fly manuelly on several trajectories, for instance back and forth. The input data and acceleration data were selected and recorded according to each model respectively. There are 2853, 2053, 2615, 2021, 3189 and 2207 data points collected in the flight experiment for model 1, 2, 3, 4, 5, 6, which represent 285.3, 205.3, 261.5, 202.1, 318.9 and 220.7 seconds flight duration respectively. The training shows that this amount of data is enough to learn the pattern and make precious prediction in this bachelor thesis.

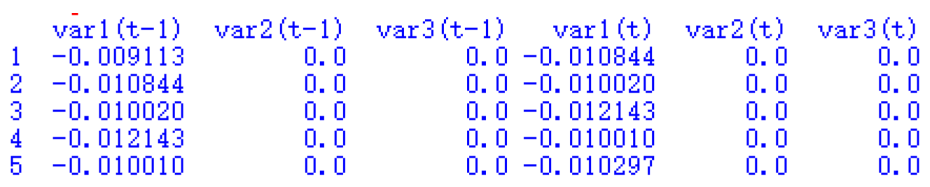
The raw data measured by CF sensor is transfered to Crazyflie PC client and generated a csv-format logdata file. In the file, each row represent a datapoint of a time point, while each column represent a measurement, which is described in the first row by its name. In the following figure, it shows the data of acceleration X and the controller input thrust and pitch in every timestamp. The time interval between two neighbor timestamps is 100ms.



	A	B	C	D	E
1	Timestamp	acc.x	controller.ac	controller.pitch	
2	90269	-0.00911342	0	0	
3	90369	-0.01084377	0	0	
4	90469	-0.01002007	0	0	
5	90569	-0.01214349	0	0	
6	90669	-0.01001012	0	0	
7	90769	-0.01029659	0	0	
8	90869	-0.01187197	0	0	
9	90969	-0.01104854	0	0	
10	91069	-0.01211369	0	0	
11	91169	-0.00927345	0	0	
12	91269	-0.01167791	0	0	
13	91369	-0.0100938	0	0	
14	91469	-0.01180311	0	0	
15	91569	-0.01106921	0	0	
16	91669	-0.01036149	0	0	
17	91769	-0.01149416	0	0	
18	91869	-0.01052469	0	0	
19	91969	-0.01184875	0	0	

Figure 8: dataset

Transformation from Sequence Data to Supervised Data In order to convert series data into supervised learning format so that it can be processed by neural network models, the collected data needs to be firstly reconstructed into a new data sets in the form of X, Y, where X is features and Y is the value that needs to be predicted, also called label. In this work, we use *lookback* to describe how many previous timesteps to be considered as features set X. The following figure shows a numpy array converted from model 1 for supervised learning, when we consider one previous time step. The first three columns shows the previous one-step measurements (acceleration X,thrust,pitch in t-1) and the second three columns shows the current data (acceleration X,thrust,pitch in t). Since the first datapoint in our series data has no previous datapoint, our converted supervised data begin with lookback-st data point in our original dataset, which means a sequence dataset with n datapoint can be transfered to be a supervised learning dataset with n-lookback data point. Take our model 1 with lookback = 1 for example, in converted data format, which are shown in figure 9, the feature set X is {var1(t-1),var2(t-1),var3(t-1),var2(t),var3(t)} and Y is {var1(t)}



```
   var1(t-1)  var2(t-1)  var3(t-1)   var1(t)  var2(t)  var3(t)
1  -0.009113        0.0        0.0 -0.010844      0.0      0.0
2  -0.010844        0.0        0.0 -0.010020      0.0      0.0
3  -0.010020        0.0        0.0 -0.012143      0.0      0.0
4  -0.012143        0.0        0.0 -0.010010      0.0      0.0
5  -0.010010        0.0        0.0 -0.010297      0.0      0.0
```

Figure 9: supervised data set

Data Normalization Because of various scales of features in this experiment, the data requires normalization. Normalization contribute to re-scaling of all features in the same range, for example in range of [0,1] (Min-Max Normalization). The purpose of data normalization is to allow neural network to learn from each features proportionately without influence of their scales. With data normalization, the performance of neural network model can be improved. Take data set in this flight experiment for example, the scale of *controller.thrust* is [0, 50625] and the scale of *controller.pitch* is [-25, 25]. With this huge difference of range of values, data normalization need to be conducted necessarily. The *MinMaxScaler class in sklearn-preprocessing* is used in this work, which transform the minimum of each feature to the parameter min and the maximum to the parameter max. Taken from [11], its mathematical expressions are:

$$X\_std = (X - X.min(axis = 0))/(X.max(axis = 0) - X.min(axis = 0)) \tag{21}$$

$$X\_scaled = X\_std * (max - min) + min \tag{22}$$

After the operation of normalization, all features in our dataset, including control input and acceleration are in the same scale and ready for training our RNN LSTM model. After training phase, we want to get the real values for prediction, the reverse operation need to be conducted for transforming normalized data into a origin scale. This operation can be done through the *inverse_transform* function in python library *sklearn.preprocessing.MinMaxScaler*.

Data Splitting Data need to be sliptted into two independent parts in artificial neural network training, called training set and testing set, respectively for the purpose of training and testing. Artificial neural network models are firstly fed with training data for fitting it, and then the trained models with updated weight and bias will be tested with test data set for model performance evaluation.

For all six models in this work, 67% of the data are used for training and the rest 33% of data for test. Because the sequence of the data point is critical in time series research, the traning data and test data are both continuous in time scale.

5.3 Modeling Architecture

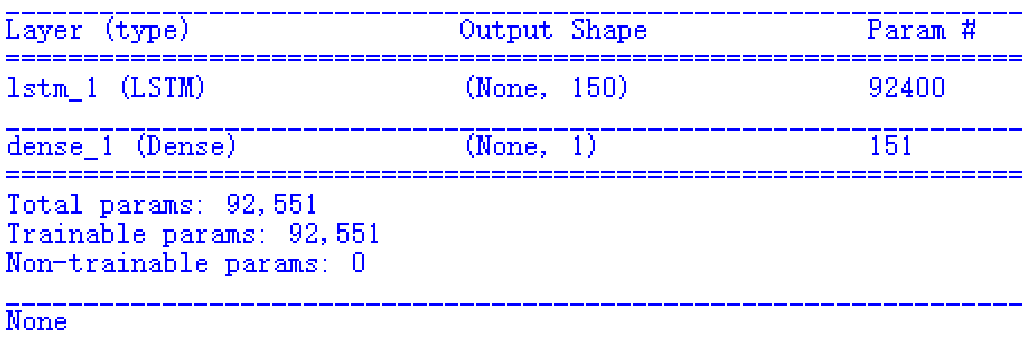
We use six preprocessed datasets collected in independent manuell flight to train six RNN SLTM models. The aim is to make six models learn from data and predict the linear and angular accelerations $\ddot{x}, \ddot{y}, \ddot{z}, \ddot{\phi}, \ddot{\theta}, \ddot{\psi}$ respectively. The target of training is minimazation of the metrics for error evaluation between real values and prediction values.

The neural network model in this work uses LSTM layer in Keras to construct, including an input layer, a hidden layer of 150 LSTM neurons, and an output layer consisting of a Dense neuron. The loss function is "Mean_Squared_Error", the optimizer is "Adam", the trained epochs is 50, and the batch_size is 72. These structure and parameters are chosen based on the work *Multivariate Time Series Forecasting with LSTMs in Keras*[11]. The parameter batch_size indicate the sample size of each training iteration. Only after the whole data in one batch_size trained, the model will then update its weight and bias between layers to minimize the loss value. One epoch means all training data are used one time for training. The structure of our neural network models are shown in format of layers and in Tensorboard[12] in Figure 10 and 11 respectively.

Metrics for Model Evaluation There are various metrics can be used for model performance evaluation. For classification, the accuracy is a common

[11]https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/, Accessed: Mai 2019

[12]https://www.tensorflow.org/guide/summaries_and_tensorboard Accessed: Mai 2019



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 150)               92400
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 151
=================================================================
Total params: 92,551
Trainable params: 92,551
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 10: Structure of LSTM model in Keras

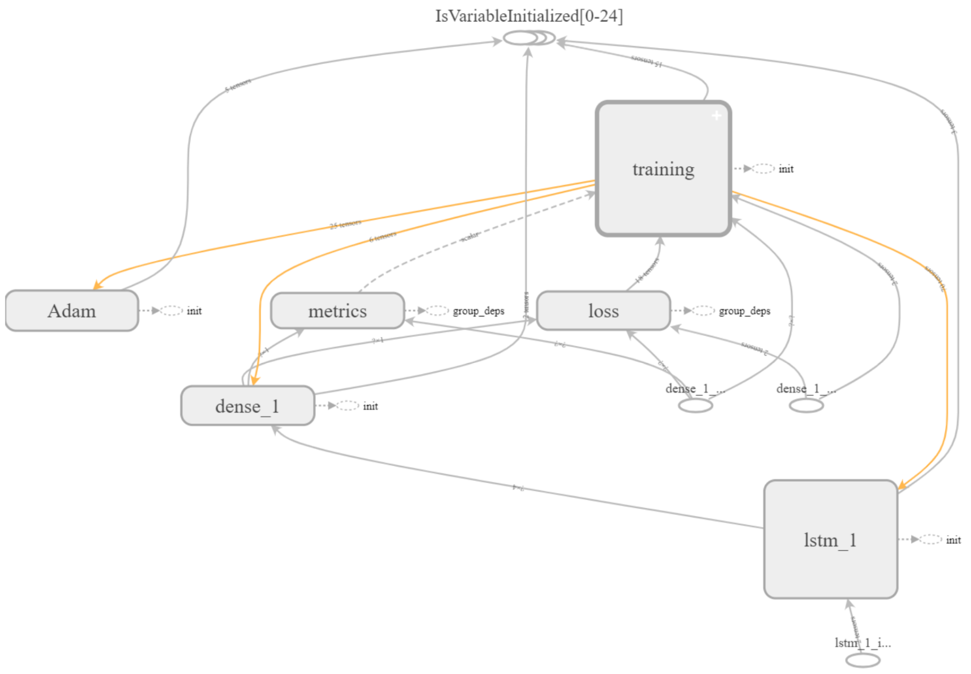


Figure 11: Structure of LSTM model in Tensorboard

metrics, which represent the percentage of correctness in classification. For regression problem the common metrics for evaluation are mean-square error (MSE), root-mean-square error (RMSE), mean absolute percentage error (MAPE), to indicate the deviation between real values and predicted values. We shoose MSE as loss function for gradient descent and RMSE for model performance evaluation in this work. The mathematical expression is:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\Big(d_i - f_i\Big)^2 \tag{23}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\Big(d_i - f_i\Big)^2} \tag{24}$$

In which, n represent the total number of the sample,i represent sample data, d_i represent the predict value and f_i is the real value.

6 Experiment Result and Evaluation

In the following pages, we show six models prediction results with their MSE change in train set and test set for each epoch, the prediction values with real values and a up-sizing graph for detailed observation. We measure the MSE with normalized data which are in range [0,1]. The result shows that, all of our six models can predict their target accelerations accurately, in which model 1, 2, 3 and 6 have better performance in compare to model 4 and 5. In model 1, 2, 3 and 6, the target accelerations are influenced by on-board stabilizer fewer than model 4 and 5 for quadcopter balance keeping. In order to keep balance, the low-level stabilizer will adjust the attitude of quadcopter automatically without control command from pilot. In model 1, 2, 3 and 6, the acceleration with lower absolute value can be more accurately predicted than those data with higher absolute value. Our assumption for this experiment result is because there are fewer train data by higher accelerations than data by lower accelerations in this work.

All of our six models are predicted by using sliding windows with lookback value equals 1 because of its best performance. Through experiments on various lookback values, we found that the best result with minimal MSE only can be achieved with lookback equals 1. It indicates that, only the flight data from one step back is important for LSTM model to learn from flight mode. With too much previous data, there are more useless information, also called noise, than useful information, so that the performance of model will be effected negatively.

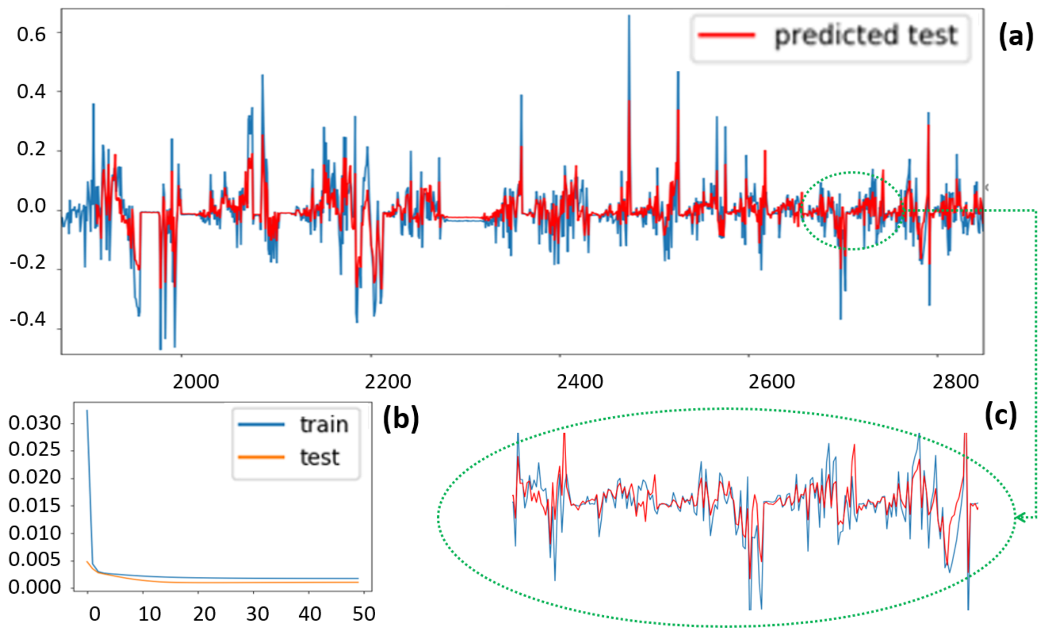


Figure 12: Prediction of Linear Acceleration $\ddot{x}$ in Model 1. In the sub-figure (a), the vertical axis scale represents the acceleration value, and the horizontal axis scale represents the data point. The predicted value is marked in red and the actual value is marked in blue. Partial enlargement of sub-figure (a) is shown in sub-figure (c). Sub-figure (b) shows the MSE changes with the epoch value in the training set and test set.

Model 1 It can be seen in sub-figure (b), as the number of training epoch increases, the normalized MSE in train set drops rapidly and maintains its value at 0.0018. After training phase, the model predict test data set and shows good performance at begin of prediction. The normalized MSE remains at 0.0011 eventually. Interestingly, the model achieve better performance using test data set than using train data set. As seen in sub-figure(a), model 1 can learn the general trend in large scale. At the same time, in

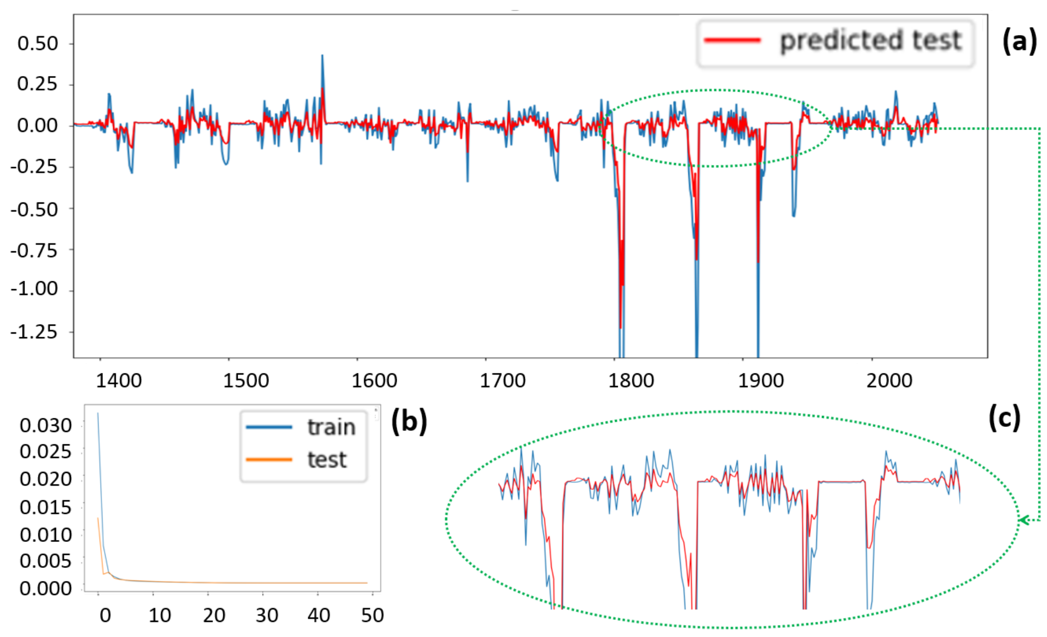


Figure 13: Prediction of Linear Acceleration $\ddot{y}$ in Model 2. In the sub-figure (a), the vertical axis scale represents the acceleration value, and the horizontal axis scale represents the data point. The predicted value is marked in red and the actual value is marked in blue. Partial enlargement of sub-figure (a) is shown in sub-figure (c). Sub-figure (b) shows the MSE changes with the epoch value in the training set and test set.

the enlarged sub-figure (c) below, we can see the detail change of the red predicted fold line following the blue actual value line. The model accurately predicts the acceleration of low absolute values, while it has poor performance in high acceleration prediction compared to the former. Our guess for this phenomenon is that the training data with high acceleration is relatively small, so the model cannot fully learn the variation pattern of high acceleration.

Model 2 In sub-figure (b), With the increase of training epoch, the normalized MSE in train set decrease rapidly and remains value at in 0.0018. After training phase, the model predict test data set and shows good performance at begin of prediction. The normalized MSE of it remains at 0.0019 eventually. Because of the good performance in the training and test data sets and the MSE in the test data set is slightly higher than the training data set, we can say that the model works very well in the acceleration prediction in this model. It can be obviously seen the capability of accurate prediction by our LSTM model which can recognize the general trend of the change of acceleration. In the enlarged graph in sub-figure (c), we can see the promptly following of red prediction fold line to the blue real value line with detail. This model predicts accurately the acceleration with low absolute value while works slightly poorly in the prediction of high acceleration. It is worth noting that, unlike Model 1, model 2 can also predict occasionally the maximum value, which indicates the effectiveness of model learning in this model.

Model 3 As shown in sub-figure (b), as the number of training epoch increases, the normalized MSE of both train set and test set drop rapidly and remain at 0.0049 and 0.0022 after ten epochs, which shows good performance of prediction. The trend of acceleration $\ddot{z}$ is simpler than the two other linear accelerations $\ddot{x}$ and $\ddot{y}$, as shown in the sub-figure (a). When the thrust input decreases, the quadcopter will descend due to gravity and the downward acceleration will be greater than the upward acceleration. The capability of our LSTM model to learn the general trend and make accurate predictions can be clearly seen in sub-figure (a). In the up-sizing graph below, we can see

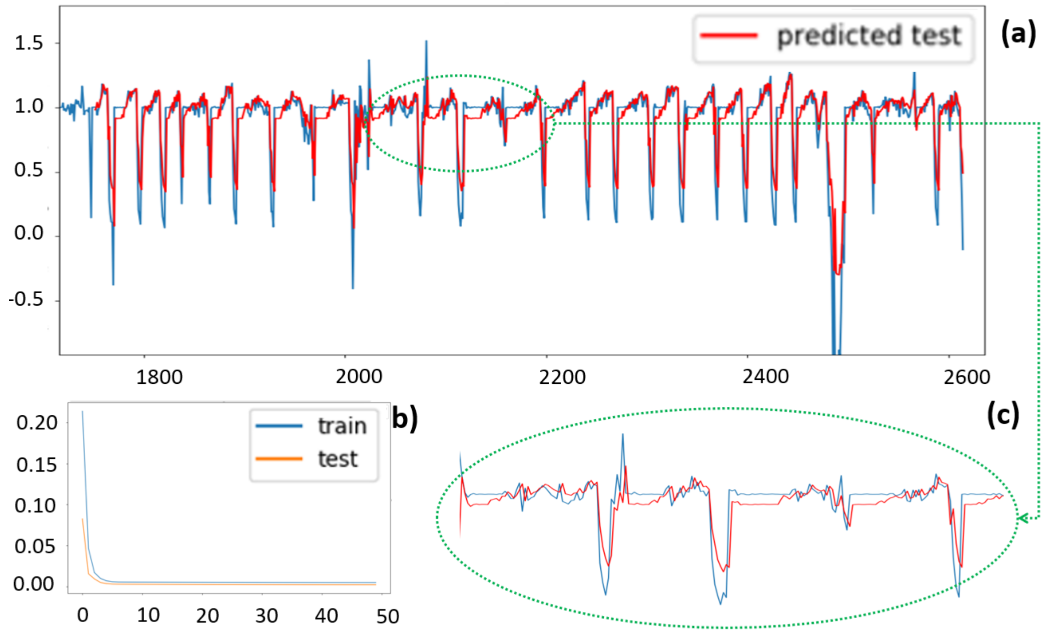


Figure 14: Prediction of Linear Acceleration $\ddot{z}$ in Model 3. In the sub-figure (a), the vertical axis scale represents the acceleration value, and the horizontal axis scale represents the data point. The predicted value is marked in red and the actual value is marked in blue. Partial enlargement of sub-figure (a) is shown in sub-figure (c). Sub-figure (b) shows the MSE changes with the epoch value in the training set and test set.

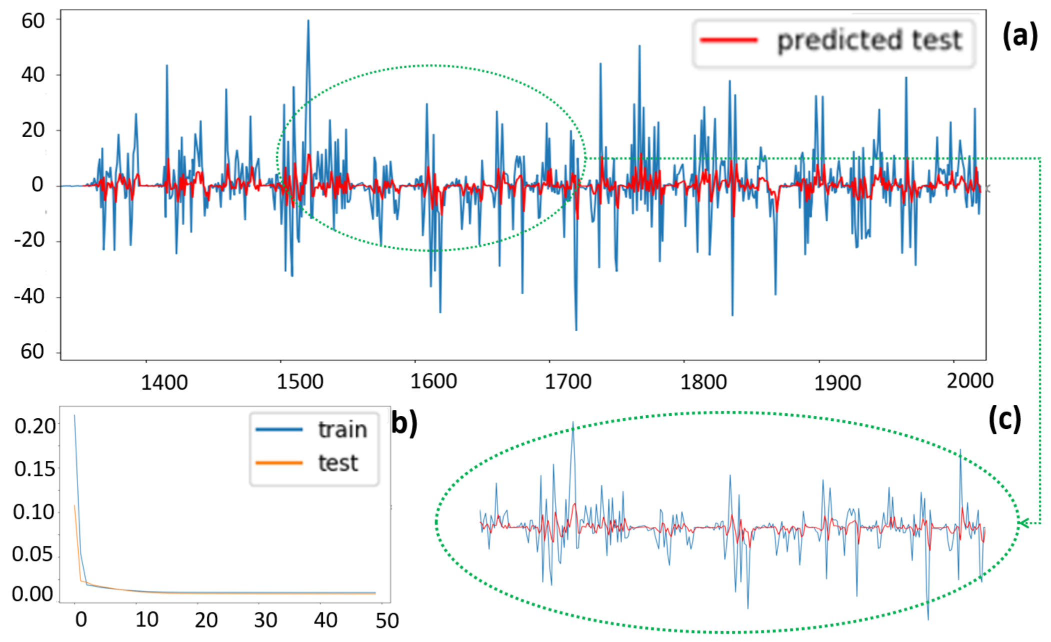


Figure 15: Prediction of Angular Acceleration $\ddot{\phi}$ in Model 4. In the sub-figure (a), the vertical axis scale represents the acceleration value, and the horizontal axis scale represents the data point. The predicted value is marked in red and the actual value is marked in blue. Partial enlargement of sub-figure (a) is shown in sub-figure (c). Sub-figure (b) shows the MSE changes with the epochs in the training set and test set.

the promptly response and following of red prediction fold line to the blue real value line. The model accurately predicts low absolute accelerations and works relatively poorly in high acceleration predictions. Since a large amount of data has a high falling acceleration, it can be well predicted and the rising acceleration of a low absolute value.

Model 4 As shown in sub-figure (b), as the number of training epoch increases, the normalized MSE in train set decrease rapidly and maintains its value at 0.0081. After training phase, the model predict test data set and shows good performance at begin of prediction. The normalized MSE stay 0.0070 in the end. The MSE in both the train and test data set is higher than model 1,2 and 3, which means it is worse than the predictions of the three models. In the part graph (a), although the red line roughly follow the blue line, some high-value accelerations could be hardly predicted. Through the study of the data set, we learned that even if the input value is zero or remains the same, sometimes the acceleration will change greatly and is higher than the normal change caused by the control input. So our conjecture is that, the extreme values in the figure may be caused by the on-board stabilizer, so our model is difficult to predict. This conjecture is confirmed by the simultaneous occurrence of bad prediction performance and high acceleration caused by non-control only models 4 and 5.

Model 5 In the sub-graph (b), the normalized MSE in train set decrease rapidly and remains at 0.0040 while the number of training epoch increases. After training phase, the model shows good performance at begin of prediction in test data sets. The normalized MSE in test data sets remains at 0.0024 eventually. As with model 4, in sub-graph (a), despite the red line roughly follow the blue line, it can hardly predict some high-value accelerations. In flight, the stabilizer always tries to adjust the attitude of the quadcopter to maintain balance. The extreme values in the figure may be caused by the stabilizer, so our model is difficult to predict.

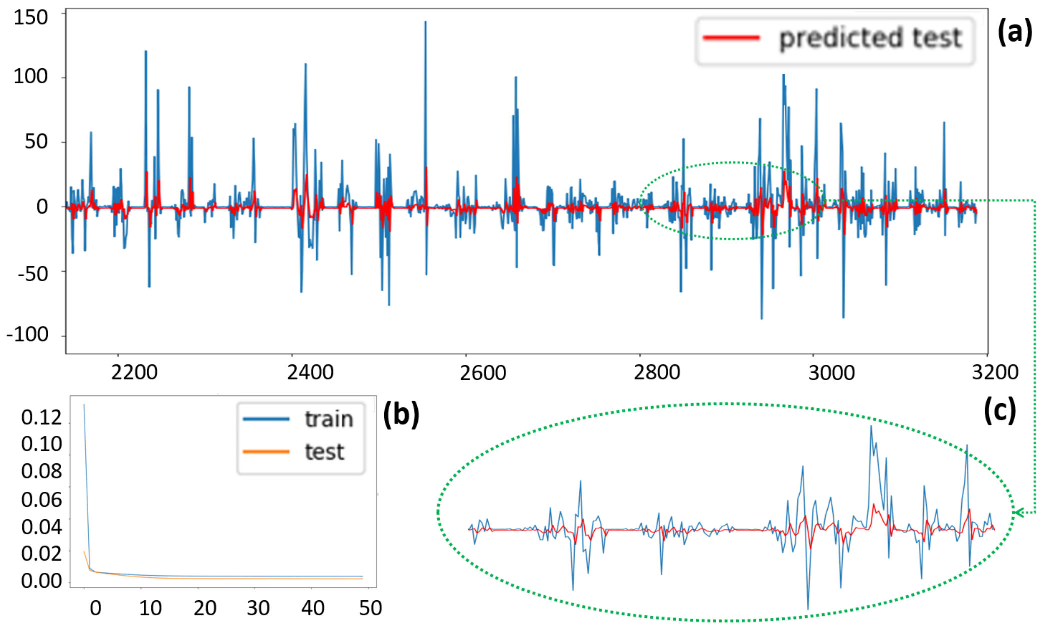


Figure 16: Prediction of Angular Acceleration $\ddot{\theta}$ in Model 5. In the sub-figure (a), the vertical axis scale represents the acceleration value, and the horizontal axis scale represents the data point. The predicted value is marked in red and the actual value is marked in blue. Partial enlargement of sub-figure (a) is shown in sub-figure (c). Sub-figure (b) shows the MSE changes with the epoch value in the training set and test set.

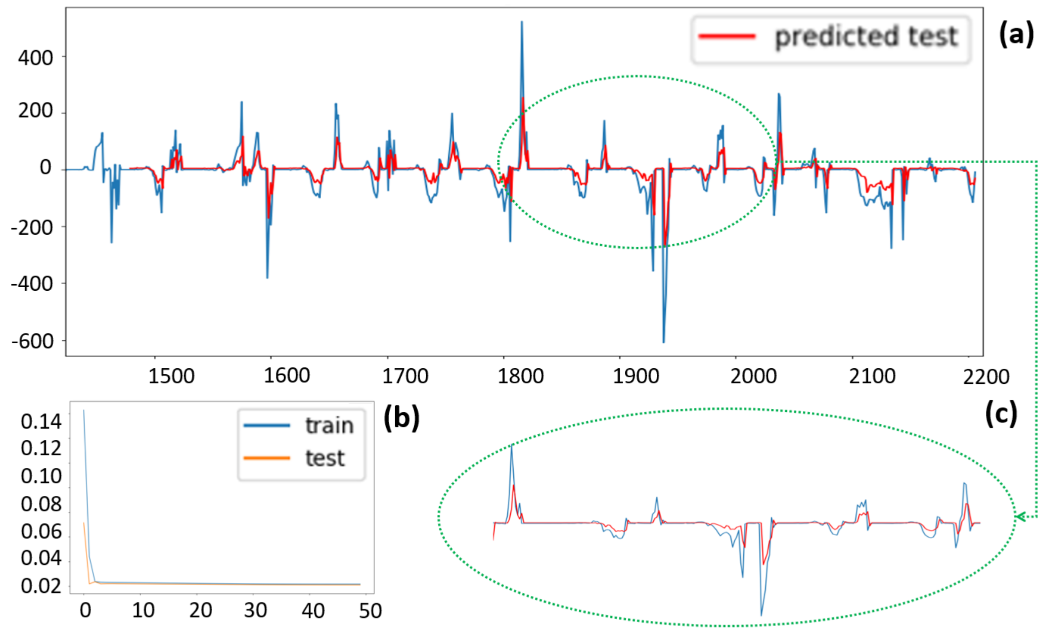


Figure 17: Prediction of Angular Acceleration $\ddot{\psi}$ in Model 6. In the sub-figure (a), the vertical axis scale represents the acceleration value, and the horizontal axis scale represents the data point. The predicted value is marked in red and the actual value is marked in blue. Partial enlargement of sub-figure (a) is shown in sub-figure (c). Sub-figure (b) shows the MSE changes with the epoch value in the training set and test set.

Model 6 It can be seen in sub-figure (b), as the number of training epoch increases, the normalized MSE in train set drops rapidly and maintains its value in 0.0017. After training phase, the model predict test data set and shows good performance at begin of prediction. The normalized MSE remains at 0.0010 eventually. Interestingly to see that the model is better using test data than using train data sets. It is clearly to see the capability of our LSTM model to learn the general trend and make accurate predictions in sub-figure (a). In the up-sizing sub-graph (c) below, we can see the promptly follow-up blue actual value line change of the red predicted fold line. The model accurately predicts the acceleration of low absolute values, while at the same time it is not effective in the high acceleration prediction compared to the former. Our assumption for this situation is that train data with high acceleration is relatively small, so the model does not adequately study this value prediction mode.

7 Conclusion

Our goal in this work is to prove the capability of quadcopters acceleration prediction by using Recurrent Neural Network (RNN) Long-Short Term Merory architecture (LSTM) based neural network model. Through experiments, the result shows the aspired goal could be successfully implemented despite limitation of some hardware resources, for example the radio bandwidth limitation. In many works, simple feed-forward neural network based models have been used for improvement of quadcopter controller design and accelerations prediction with full state data of quadcopter. In compared to those models with fully neural network modeling with state data of quadcopters, including its linear and angular velocity, space position and attitude, we only use partial collected data of control inputs and state data for model training in this work. In the experiment, we shows that RNN LSTM model can also be used for making precious prediction, except two angular accelerations prediction with relative low performance among six models. By introducing the time-series concept in dynamic model training, not only the dependence of control input and acceleration but also the dependence of acceleration in time sequence are taken into consideration. With the previous input and acceleration data, the RNN LSTM model is capable to estimate its state and combine the previous state with current input data to make precious prediction. Although this RNN LSTM model shows ability of quadcopter dynamic modeling processing in this experiment, there are also limitation and flaws of prediction, for example not accurate enough or no recognition of some flight pattern, especially in model 4 and 5, it can not recognize the acceleration caused by stabilizer. To improve the performance of the neural network, the future work can be on the one hand, doing research of RNN LSTM model implementation with other RNN architechture, for example the Gate Recurrent Unit and compare this performance with LSTM model. On the other hand, enlarging the training dataset and improving the quality of training data are also two promising ways for a better prediction performance. To improve the quality of training data, we can drop some noise data out of the training data set, and use the flight data collected during more meaningful trajectory or more stable environment, which can be achieved by autonomous flight using code control.

8 Bibliography

References

[1] 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).*

[2] N. O. Lambert, D. S. Drew, J. Yaconelli, R. Calandra, S. Levine, and K. S. J. Pister, "Low Level Control of a Quadrotor with Deep Model-Based Reinforcement Learning", Department of Computer Science and Robotics, Cornell University, in arXiv:1901.03737v1 [cs.RO], 11 Jan 2019.

[3] W. S. McCulloch and P. Walter, "A logical calculus of the ideas immanent in nervous activity", in *The bulletin of mathematical biophysics 5*, N. Rashevsky, pp. 115–133, 1943.

[4] T. Luukkonen, "Modelling and control of quadcopter", School of Science, Aalto University, Espoo, August 22, 2011.

[5] F. A. Gers, J. Schmidhuber, F Cummins, "Learning to Forget: Continual Prediction with LSTM" *9th International Conference on Artificial Neural Networks: ICANN '99* p. 850 – 855, 1999.

[6] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, G. M. Ljung. *Time Series Analysis: Forecasting and Control* Holden-Day, Inc. San Francisco, CA, USA, 1990.

[7] P.S.P. Cowpertwait, *Introductory Time Series with R*, Springer, 2009.

[8] A. Graves, “Supervised Sequence Labelling with Recurrent Neural Networks”, Ph.D. dissertation,Fakultät für Informatik, Technischen Universität München, 2012.

[9] R. Jozefowicz, w. Zaremba, I. Sutskever. “An empirical exploration of recurrent network architectures”, in *International Conference on Machine Learning - Volume 37*.p.2342-2350, 2015.

[10] W. Zhu, C. Lan, J. Xing, W. Zeng, Y. Li, L. Shen, X. Xie. “Co-Occurrence Feature Learning for Skeleton Based Action Recognition Using Regularized Deep LSTM Networks”, in *AAAI’16 Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, Phoenix, Arizona — February 12 - 17, 2016* . p.3697-3703, 2016.

[11] F. Pedregosa, G. Varoquaux et al., “Scikit-learn: Machine Learning in Python” in *Journal of Machine Learning Research,* JMLR 12, pp. 2825-2830, 2011.

[12] A. Graves, A. Mohamed, G. Hinton “Speech Recognition with Deep Recurrent Neural Networks”, in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings. 38. 10.1109/ICASSP*, 2013

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

[14] R. J. Williams and D. Zipser, “Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity”; in *Back-*

propagation: Theory, Architectures and Applications, Y. Chauvin and D. E. Rumelhart, Eds. Hillsdale, NJ: Erlbaum, 1995, pp. 433–486.

[15] N. Mohajerin, "Modeling Dynamic Systems for Multi-Step Prediction with Recurrent Neural Networks", Ph.D. dissertation, Department of Mechanical and Mechatronics Engineering, University of Waterloo, 2017.