A Recurrent Neural Network for Nonlinear Control of a Fixed Wing UAV

A Thesis

Presented to the

Faculty of

California State Polytechnic University, Pomona

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

In

Computer Science

By

Ohanes Dadian

2017

SIGNATURE PAGE

THESIS: A Recurrent Neural Network for Nonlinear
Control of a Fixed Wing UAV

AUTHOR: Ohanes Dadian

DATE SUBMITTED: Summer 2017
Computer Science Department

Dr. Amar Raheja
Thesis Committee Chair
Computer Science

Dr. Daisy Tang
Computer Science

Dr. Subodh Bhandari
Aerospace Engineering

ACKNOWLEDGEMENTS

There are many people and groups I would like to thank, whose support were invaluable in this effort.

I would like to thank Dr. Raheja, for his guidance throughout my college education. It has been a pleasure doing research with him from my senior project to my master's thesis. He instilled in me the importance of understanding the theoretical underpinnings of the problem you are solving. Only then can you achieve an application for the problem that is not only sound and complete, but optimal as well.

I would also like to thank Dr. Bhandari. His guidance on flight dynamics and control was invaluable. He is, perhaps, one of my favorite professors at Cal Poly. Ironically, I have not taken a single class with him. In the time I got to work with him, we built quite a kinship. He helped be achieve the type of graduate school experience I didn't know was possible. Publishing papers and presenting at conferences all over the country.

One person that deserves a special mention is my research partner, Ajay Bettadapura. He was my aerospace counterpart throughout this entire endeavor. He consistently inspired me to attack the most interesting and challenging problems in unmanned space. From the recurrent neural net to the modular avionics framework, I feel we have put many new toys into the unmanned systems toy box.

This research effort could not have been possible without the National Science Foundation. Their funding was the life blood of our research group.

In addition, I would like to recognize Hovig Yaralian for performing flight tests through the duration of our trials. His tireless efforts provided us with the experimental results we needed to validate our algorithms.

Last, but certainly not least, I would like to thank my parents. Avedis and Hripsima Dadian have been the most wonderful role models a child could ask for. They constantly inspire me. At a young age, they instilled in me the need to not only succeed but do good in this world. To leave it better than I found it. For that, I am eternally grateful.

ABSTRACT

As the pursuit for developing next-generation flight control systems continues, so does the need for intelligent agents which are capable of providing increased autonomy and stability while ensuring safety. Many different models have been utilized in the past, such as gain scheduling. However these techniques tend to incur instability when dealing with varying weather conditions. This is due to their inability to handle nonlinearity in the model.

To solve this problem, a nonlinear model is required. One that can be trained offline and adapt in-mission. There are many models to perform the parameter tuning involved for providing stability to aircraft.

This thesis showcases one such application, utilizing a recurrent neural network for providing intelligent nonlinear control. Funded by the National Science Foundation (NSF), the Computer Science and Aerospace Engineering departments at Cal Poly Pomona have been conducting an effort to develop adaptive flight controllers for unmanned aerial vehicles. The current focus is on an echo state network for performing nonlinear control. It was chosen due to its close adherence to classical control law.

An introduction to neural networks and echo state networks is given to provide context of the subject matter. This is followed by a literature survey of current implementations of recurrent neural networks in flight control systems. A discussion is then given on our research goals, focusing on the echo state network and its role within our intelligent flight control system (IFCS). The methodology used to accomplish these goals is then given; both the MATLAB-based prototype and the C++ implementation. The closed-loop simulation system is detailed, including offline and online training using the FlightGear flight simulator and Athena Vortex Lattice (AVL) flight dynamics analyzer. The thesis concludes with evaluation of results generated during software-in-the-loop and flight testing.

TABLE OF CONTENTS

Signature Page	ii
Acknowledgements	iii
List of Figures	vii
1 Introduction	1
2 Literature Survey	8
3 Research Goals	15
3.1 Objectives	16
4 Methodology	19
4.1 System Identification	19
4.2 Neural Network	20
4.2.1 Model	20
4.2.2 MATLAB Prototype	23
4.2.3 C++ Implementation	23
4.2.4 Closed-Loop System	24
4.3 Simulation	25
4.4 Platform	28
4.5 Aircraft	28
4.5.1 Flight Dynamics Model	29
4.6 Flight Lab	31
5 Evaluation of Results	33
6 Conclusion and Future Work	48
7 Glossary	49

8	Source Code	52
9	References	90

List of Figures

1.1	Recurrent Model	6
1.2	Echo State Network	6
2.1	Elman Model	9
2.2	Jordan Model	10
2.3	Hybrid Jordan Model	11
2.4	Blackbox Network	12
4.1	Twin-Engine Flight Dynamics Model	20
4.2	Offline Model	22
4.3	Hyperbolic Tangent	24
4.4	Closed Loop	24
4.5	FlightGear Simulation	25
4.6	FlightGear Generic Protocol	28
4.7	Parallel Board	29
4.8	Twin Engine Aircraft	31
4.9	Flight Lab	32
5.1	Twin Engine Aileron Input Command	34
5.2	Plant Input and Output	35
5.3	Commanded Bank Angle	37
5.4	Online Response with Gusts	37
5.5	Response Comparison of Online and Offline Controller	39
5.6	Inputs and Desired Output to Offline Neural Network	39
5.7	Simulated Roll Rate	40
5.8	Aileron Input Command	41
5.9	Roll Rate Output	42
5.10	ESN Roll Rate	43
5.11	Pitch Rate	44

5.12 ESN Online Model Response	44
5.13 Commanded Bank Angle	45
5.14 Closed Loop Response for Bank Angle	45
5.15 ESN Bank Angle Response	46
5.16 Online Response with Gusts	47
5.17 PID Tuning with ESN and MLP	47

1. INTRODUCTION

Over the past decade, Artificial Neural Networks (ANN) have grown into prominence. These models have been used to solve various problems in science and engineering. This includes, but is not limited to, character recognition, computer vision, time series prediction, signal processing, and autonomous control among others. ANNs provide nonlinear learning to classically linear programmed systems. Allowing such systems to acquire skills that previously required human operation and insight. These intelligent agents are software-based computational models, which emulate the learning patterns found in human central nervous systems. Computational nodes, or neurons, are connected in series to build out a network graph. The connections between the neurons are weighted, as to ensure a critical path of learning is established. The neurons hold a value generated by an activation function, providing nonlinear approximations of their input. In the case of recurrent models, the activation would also be driven by the output signal; as it is driven back into the model.

Computational systems have long been inspired by the learning behavior of the human brain. A classic example of this is the paper by Warren S. McCulloch and Walter Pitts, "A logical calculus of the ideas imminent in nervous activity". [37] This publication, released in 1943, described the first computational model of an artificial neural network. At the core of the model, lies an artificial neuron; a single component interconnected with others to form a network. Akin to biological neural networks, where a single cell lives in a network of cells. Like a biological cell, it processes input and generates output. The intent of the model was to provide a means of nonlinear learning for computing systems.

The concept of the artificial neural network was born out of desire to overcome the difficulty computers had with a certain subset of problems, which come naturally human. For instance, a person can be given a picture and easily recognize the object in frame. However, a computer will have an exponentially difficult time performing object recognition, depending on the noise in the frame. In Computer Science, these sets of problems are referred to as pattern recognition.

The artificial neural network, as described in the paper, was a connectionist model that

attempted to solve this deficiency. Unlike procedural systems, connectionist models follow a nonlinear path of computation. This suggests that computing is performed in a parallel fashion, with results collected over time. Providing an adaptive capability for learning skill in particular domains. Adaptability allows for change in a model's internal structure based on the input signals driven into it. This is accomplished through the adjustment of weights, which dictate the strength of connections between neurons. Once the network is capable of generating an output signal that is acceptable within a given delta, no further weight adjustment is required.

Within the aerospace industry, research and development of unmanned aerial vehicles (UAV) has become increasingly popular. UAVs are the fastest growing sector within this space. Their rise to prominence may be attributed to the cost and safety effectiveness of their applications. UAVs are capable of performing missions that would usually require a human pilot. Taking the human out of the loop allows for increased safety. The cost of UAVs are significantly cheaper, as single unit and operating costs are greatly reduced. They can fly to places that are impossible for manned aircraft to fly to. Despite these many advantages, the use of UAVs are limited to restricted and military airspace. In order to be able to use these vehicles as a viable replacement for manned aircraft for number of missions described above, the development and operating cost must be reduced while increasing availability, reliability, robustness, and situational awareness. Increased reliability and robustness will enable them to complete their mission requirement without loss in the event of hardware and software failure and in the event of external disturbances. In order to utilize their full potential, the UAVs must be integrated into National Air Space (NAS). This requires sense and avoid capability and increased autonomy.

UAVs are used for performing various military and civilian missions; including battlefield damage assessment, search and rescue, reconnaissance, and forest fire detection. In order for these aircraft to be used within the varying missions discussed, it is apparent that they will require flying within a wider flight envelope. This would require UAVs to exhibit a high level of autonomy in order to be registered within the National Air Space. To provide this capability, their flight controllers would need to account for varying flight conditions and nonlinearities. The rise of UAVs is expected to increase in the future.

However, despite continued effort on increasing the robustness and autonomy of these vehicles, the UAVs are not yet robust enough for integration into the National Air Space (NAS), thereby limiting their operation in the restricted airspace. One of the main reasons for the lack of robustness is the use of controllers that are designed using classical control system design techniques. Such controllers are based on linearized flight dynamics model and are designed to work for particular flight conditions. If the flight dynamics changes due to parametric uncertainties in the model, due to external disturbance such as wind gust and turbulence, due to changes in flight speed and/or altitude and mass, the controllers most likely will not function. The controllers will also not function when the configuration of the vehicle changes due to component failure. This will result in the loss of the vehicle and expensive onboard sensors, equipment, and information stored. For increased robustness, reliability, and availability, due to absence of human pilot, UAVs must be able to autonomously control themselves in the situations described above. One might ask themselves, how would such a neural network perform this learning process? In fact, there are various techniques usually employed; supervised, unsupervised, and reinforcement learning. In supervised learning, a teacher signal is used to train the model to generate an actual output within a close approximation of the teacher output. Unsupervised learning provides a means to find a solution when a sample set doesn't exist. In Reinforcement Learning, the outcome of intermediate decisions are analyzed to assess its success. Weights are adjusted based on this assessment.

There are many control techniques that can be employed to achieve this capability; including nonlinear dynamic inversion and gain scheduling. However, these techniques require a priori knowledge. This caused a need for a control system that does not require current knowledge. This lead to a focus on neural network, which could possibly provide this capability. These models would allows approximating nonlinear functions to a high degree of accuracy, using input-output pairs collected during flight tests. There has been much work on the application of neural networks for flight control. However, much of this research focused on feed-forward models. Unfortunately, they do not naturally adhere to classical control law. This is due to their lack of a feedback loop. The design would inherently have difficulty exhibiting adaptability.

Gain scheduling is a popular method for handling parametric uncertainties and disturbances [11,12]. Gain scheduling works by linearizing the flight dynamics model at several operating conditions and controller are then designed for each linearized model. These controllers are then arranged in a look up table and are used when the operating conditions are detected. Gain scheduling has been used quite extensively in aircraft control. However, gain scheduling has many disadvantages. Frequent and rapid changes of controller gains may lead to instability [13]. Another disadvantage is the high design and implementation costs that increase with number of operating costs. Controller gains are pre-computed off-line and the mechanism provides no feedback to the incorrect schedules. Incorrect prediction in operating points may lead to vehicle failure. Some of these disadvantages can be overcome by using nonlinear controller based on feedback linearization. A common feedback linearization technique used on aircrafts is dynamic inversion. However, dynamic inversion may not work if the dynamic model is inaccurate or the system is non-minimum phase. To overcome this issue dynamic inversion is used alongside a robust nonlinear controller. These robust controllers are only good for some bounded uncertainties and cannot handle unbounded uncertainties or unmodeled dynamics. To overcome these issues, UAVs need onboard algorithm for the calculation of controller parameters that take into account disturbance, parameter uncertainties, modeling error, and unmodeled dynamics. Nonlinear adaptive controllers can help UAVs overcome these issues. Adaptive controllers help the UAVs adapt to different situation. They have the potential to control the vehicle even in the occurrence of component failure. Many universities have been doing research on the design and implementation of adaptive controllers. With few exceptions, most of these works, however, are only theoretical derivations or are limited to numerical or computer simulations. Similar simulation and theoretical results have also been shown for UAVs. Some limited works exist on the flight testing of adaptive controllers. However, not all the adaptive controllers are robust. Adaptive controllers can easily go unstable or do not perform as expected in the presence of parametric uncertainties, unmodeled dynamics, and bounded disturbances. To overcome this issue Artificial neural networks (ANNs) can be used alongside an adaptive controller. Artificial neural networks (ANNs) are a class of non-parametric models based

on the neuron interaction in the human brain. ANNs can approximate essentially any function with arbitrary accuracy if provided with sampled data from the system. The function is derived directly from the sampled data. Many different forms of neural network architectures exist, each of which is suitable towards various tasks of prediction, classification, pattern recognition etc. The most commonly used network is the multi-layer perceptron (MLP) feed forward network with some form of error back propagation as the learning algorithm. The other popular but lesser used neural network architecture is the Radial Basis Function (RBF) neural network. ANNs have the ability to approximate continuous nonlinear functions and are ideal for the adaptive flight control application. Its nonlinear nature provides the system with flexibility to achieve practically any desired input/output mapping. Neural networks function as highly nonlinear adaptive control elements and offer distinct advantages over more conventional linear parameter adaptive controllers in achieving desired performance [57]. We plan to use MLP neural networks that can be trained a-priori with controller inputs which would be collected as real flight data or simulated to provide a larger training data set covering a wide range of possible input values to the controller. The work of Rysdyk and Calise investigated architectures that provide neural network augmentation to a model inversion controller. In our case, we will be working with offline trained as well as online neural networks for adaptive control design [59]. Offline neural network approximately inverts the vehicle dynamics. Online adaptive learning neural network can then compensate for nonlinearity and inversion error and thus improve the performance of the controller. The ANNs are also robust in that when an element of the network fails, it can continue because of their parallel nature. A stable weight adjustment rule for the online neural network will be derived using a Lyapunov stability theorem. The Lyapunov proof will also guarantee the boundedness of the tracking error.

The goal of this thesis is to design a recurrent neural network, which will provide nonlinear control of a fixed wing UAV. The recurrent nature of the model will allow for strict adherence to classical control law and more easily provide adaptability. Many recurrent models were considered. However, the Echo State Network was ultimately chosen.

Figure 1.1: Recurrent Model

Figure 1.2: Echo State Network

Echo State Networks (ESN) provide a novel reservoir based approach combined with supervised learning. Very practical and computationally inexpensive by design, ESNs provide a means to generate signals based on current knowledge throughout the model history, and utilizes a leak rate for this current knowledge in the hidden layer. At the heart of its simplistic design is the sparse connectivity between its neurons. This allows for a leaner model and less complex computations in generating the output signal. These design choices provide an overall architecture that is simpler and easier to implement than the aforementioned models. As with the other recurrent models, the output neurons are driven back into the model.

2. LITERATURE SURVEY

Neural Networks have become a popular topic of research among Computer Scientists in academia. Perhaps one of the most popular applications of this intelligent agent is its use in controlling Unmanned Aerial Vehicles, or UAVs. Cal Poly Pomona has begun looking into various neural networks for use in adaptive control of UAVs. As this effort has already spent time assessing feed-forward networks, recurrent networks seemed like the logical next step. The preceding work is a comparison and evaluation of said networks. Neural Networks are ever increasingly being applied to many military and civilian applications. One such application is that of flight control, particularly of unmanned aerial vehicles. Many different implementation of neural network classes have been utilized within flight control research.

A group at Universidad Politecnica de Madrid in Spain developed a hybrid supervised approach. [19] They combined Elman and Jordan networks. Elman networks, developed by Jeff Elman, is a three layer recurrent model. They contain context units that connect the neurons within the hidden (middle) layer. Training occurs at each time quantum, with back propagation. In addition to this weight adjustment, a learning rule is applied to the model. The Elman model is represented in equations (2.1) and (2.2). The input vector is denoted by x_t . While the hidden layer vector is denoted by h_t . This leaves the output vector, which is denoted by y_t . W , U , and b are parameter matrices and vectors. The activation functions are shown as δ_h and δ_y . The learning rule provides context units with copies of the previous values within the hidden layer. This allows Elman networks to lend themselves to dynamical systems. An adaptation of the Elman network was created by Michael I. Jordan. In the Jordan network, the context units are driven into the output layer, instead of the hidden layer. The units also have a recurrent connection to themselves. The Jordan model is represented by equations (2.3) and (2.4). The chosen design was elaborated into a dual block design. Block A holds a feed-forward model, while Block B represents a recurrent model. This can be seen in figure 2.3. The input vector is defined as $X = [x_1, x_2, x_3, \dots, x_n]$. This generates context neurons C_{xi1} to C_{xid} . The output vector is formulated as $Y = [y_1, y_2, y_3, \dots, y_n]$. This in turn generates context neurons C_{yi1} to C_{y1h} .

Figure 2.1: Elman Model

The hybrid network works with a series of previous states. Those previous states are used to compute the X input vector, by multiplying the previous states d by the input states x . The h previous states are multiplied by the output states m to compute the output vector. They required two networks, one for the radio signals to attitude system and another for the attitude to position system. Two methods of training were tested; daisy chained and decoupled. The daisy chained architecture uses Block B to train Block A. This was contrasted with the decoupled architecture, which treated the both blocks as isolated systems. They tested the models with both real and simulated data. The benchmarks proved that the daisy chained architecture provided better performance. It clearly had a lower overall mean square error throughout all sessions. This neural network provides a great proof-of-concept for Cal Poly Pomona's research effort. Their work shows that aspects of different architectures can be taken and used to build a hybrid system.

$$\mathbf{h}_t = \delta_{\mathbf{h}}(W_h x_t + U_h h_{t-1} + b_h) \quad (2.1)$$

Figure 2.2: Jordan Model

$$\mathbf{y}_t = \delta_y(\mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y) \quad (2.2)$$

$$\mathbf{h}_t = \delta_h(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h \mathbf{y}_{t-1} + \mathbf{b}_h) \quad (2.3)$$

$$\mathbf{h}_t = \delta_y(\mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y) \quad (2.4)$$

Some have attempted the use of autoregressive neural models for flight control. This provides the behavior of computing future values based on the weighted sum of past values. Past values consist of the input and output signals. Providing recurrent behavior, such as autoregression, allows for seamless application of dynamical systems. As this gives the model equivalent retention capabilities of nonlinear dynamics within the system. University of South Wales developed a network which uses a black box model for system identification and estimation. This can be seen in figure 2.4. This system provides adaptive control via identification and control blocks. This control system is supposed to overcome disturbances and dynamics stemming from the environment. Their proposed

Figure 2.3: Hybrid Jordan Model

system is comprised of two network models. One of which is a trained online, in order to adapt to changes to the environment. The proposed network is supposed to provide fairly good retention, as it uses past inputs and outputs along with current inputs. The network has been tested through numerical simulation, real time validation and varying flight missions. The output signal is generated by a two phase computation of the hidden and output layers. This can be seen in equation (2.5). W_{2js} and W_{1ij} are the weights between the connected neurons. W_{20} and W_{10} are biases. Indices of the input, hidden, and output layer are represented by i , j , and s . The actual input signal is denoted by x_i . F_s and G_j are the activation functions of the neurons. While l_1 and l_2 are the sizes of the input and hidden layers respectively. The network is equipped with a modified version of Levenberg-Marquardt training algorithm, which governs the online training. It also includes two feedback loops that are supposed to improve overall accuracy of the system. Levenberg-Marquardt is a gradient based iterative training technique which attempts to minimize an error function. It minimizes the mean square error formula as a function of θ . This can be seen in equation (2.6). Function $y(t)$ is the target output and $g(\theta, \phi(t))$ is the neural network predicted output optimized by Levenberg-Marquardt.

Figure 2.4: Blackbox Network

$$\mathbf{y}_s(\mathbf{t}) = F_s \left(\sum_{j=1}^{l_2} W_{2js} G_j \left(\sum_{i=1}^{l_1} W_{1ij} x_i(t) + W_{10} \right) + W_{20} \right) \quad (2.5)$$

$$\mathbf{V}_N(\boldsymbol{\theta}) = \frac{1}{N} \sum_{t=1}^N (y(t) - g(\boldsymbol{\theta}, \phi(t)))^2 \quad (2.6)$$

The Naval Aeronautical Engineering Academy has conducted neural network research to provide control of tight formation flight with minimum power reduction for multiple UAVs. This capability is to be provided by their proposed neural network. At the heart of this network is the sliding mode control and Extremum Seeking Algorithm (ESA). The sliding mode control allows modification of the system dynamics via discontinuous control signals. This component is what allows for control of flight formation. Conjointly, the ESA algorithm may be seen as their form of adaptive control. It is meant to maximize the aileron reflection of the UAV in order to cause a significant power reduction. The algorithm was validated using simulation and convergence analysis. Their testing proved that the ESA algorithm allowed for an optimal convergence of flight formation with a power reduction of 40 percent. This is 20 percent greater than their projected reduction

during simulation.

Since many UAVs employ finite state machines, it may prove useful to have some fundamental analysis on building finite state automata on recurrent neural networks. Perhaps the most promising research project has to be one out of Princeton. Christian Omlin designed a deterministic finite state automaton that produces a second-order recurrent neural network. The issue with implementing neural nets as automata is that their inherent behavior exhibits deteriorating results on long strings. The strings being the input; whatever that may be for the domain involved. Perhaps the two leading causes of this deterioration are the internal representation of the automaton and the sigmoid function. Omlin set out to alleviate said issues with his proposed deterministic finite state automaton (DFA). This DFA produces a network with a second-order network with sparse interconnection topology and a sigmoid function with stable DFA states. This algorithm produces a network with $O(n)$ neurons and $O(mn)$ weights. From a theoretical standpoint, this looks to be quite impressive. It might prove to be quite helpful when fitting the experimental model of our recurrent net.

Echo State Networks have been gaining popularity as of late. This is a class of recurrent neural nets with a sparsely connected hidden layer. An interesting implementation of this architecture is one out of Tsinghua National Laboratory. They built a hybrid system combining an echo-state network with a state-space angular motion model. The echo-state net was designed as a black-box and solves the problem of modeling for horizontal translational motion. The hybrid system effectively provides four-degrees of freedom for a UAV. In the case of this research project, the UAV being used is a Raptor 30 helicopter. The identification control is fed data from the remote controller in order to establish online training. The motion modeling was proved via validation results. Further validation was provided through closed-loop simulation. This network provides a fascinating approach to recurrent nets for UAVs. Up until now, standard recurrent models implemented. Having only one percent of connectivity in the hidden layer provides a much simpler model to implement and fit. Although it is nonlinear, the error term is quadratic and can be easily characterized as a linear system. It is also beneficial that they implemented the network on a raptor helicopter, since Cal Poly Pomona employs one of its own for UAV research.

This provides a great starting point for performing flight control research with echo state nets.

Technische Universität Braunschweig has developed a neural network with an observer based pattern. They identified two crucial issues for providing adaptability in flight; environmental conditions and system failure. The consensus lead to neural networks as desired control system, as their nonlinearity aids in overcoming such uncertainty. An offline and online model were employed. The offline model used a feed-forward approach, while the online model utilized an observer pattern. The observer models system dynamics and provides quality signal for training. Error determination is accomplished by backpropagation. The training algorithm was derived from sliding mode control. This is a non- control method that use discontinuous signals to provide a variable cross-section of system behavior, through discrete control structures. The discontinuous behavior allowed for improved characteristics for the observer pattern. As the observer model stabilized, the neural state was able to achieve gains in the adaptability of the online model. A Carolo P200 aircraft was used for flight testing. On board sensors were used to measure dynamics and pressure of the aircraft. A Kalman filter was used during error computation, for noise reduction. Lateral control was achieved by ailerons and elevators, with no rudders. Flight attitude is computed beforehand for every point trajectory. Deviation from spline was computed at every control step. The attitude and deviations were used to inform the input and output signals. The Levenberg-Marquardt method was used during back propagation. Basic knowledge was acquired during offline training, with adaptation occurring during online flight.

3. RESEARCH GOALS

The aim of this research effort is to design and implement a neural adaptive flight controller for nonlinear control of a fixed wing unmanned aerial vehicle. To reach this objective, two methods have been considered. The first was a feedforward-based model, while the other was a recurrent-based model. For the feed-forward model, the multi-layer perceptron was used. It consists of a three layers; an input, hidden, and output layer. The input layer was driven by PID values from the twin-engine. The network is trained using a teacher signal formulated by flight data. Weight adjustment of the neuron in the hidden layer is accomplished by backpropagation. For the recurrent model, an echo state network was used. It also consists of an input, hidden, and output layer. The hidden layer contains a 40 percent leak rate of current knowledge and 10 percent connectivity between neurons. The output layer is activated using Weiner-Hopf based linear regression. Weight adjustment is accomplished by back propagation through time. This unfolds the temporal state spaces of the network and recursively performs backpropagation through the model at all time quantum within the training epoch.

The recurrent network, by design, more succinctly adheres to classical control law. Through its intrinsic dynamical behavior it is able to model a flight control system; complete with a feedback loop. This has caused us to go with this model as our controller, with the PID feedback from the Twin Engine as the plant. The teacher signal for the network is comprised of the PID feedback generated from a flight test of the twin engine. The PID feedback loop takes the error term between the plant and control system of the flight controller and adjusts it according to proportional, integral, and derivative terms. This effectively minimizes the error of nonlinear control for the aircraft. The input signal will be formulated with the roll, pitch, and yaw data stemming from the flight test. The echo state network will train against the flight data until the dynamics governed by roll, pitch, and yaw has been tuned within an acceptable delta of the plant (PID feedback).

3.1 OBJECTIVES

DEVELOP AND VALIDATE NONLINEAR FLIGHT DYNAMICS MODEL

A nonlinear model is required to formulate the input-output pairs for governing the dynamics of the Twin-Engine aircraft. The simulated roll, pitch, and yaw data sets will allow for predicting the behavior of the aircraft. The data sets are compiled out of three force and three moment equations used to describe the rigid body motion of the aircraft. These equations are reformulated with stability and control derivatives for better compatibility with our control system design.

1. Develop flight dynamics model
2. Reformulate with control and stability derivatives

DEVELOP RECURRENT NEURAL NETWORK FOR NONLINEAR FLIGHT CONTROL

The neural network will lie at the epicenter of our flight control system (FCS). Playing the part of the PID tuner within the feedback loop. The network will employ 17 neurons in its reservoir with 10 percent connectivity and a 40 percent leak rate. Simulated flight dynamics data (roll, pitch, and yaw) will be compiled into input-output pairs and used for offline training. Supervised learning will continue until the output signal reaches an acceptable delta with the simulated teacher signal. Once the model has reached a fitted state, it can then be placed within the control system taking the PID values from the PID controller, tune them, and feed them back into the controller. The objectives can be formulated as the following.

1. Develop neural network model with 17 neurons and 10 percent connectivity
2. Feed input-output pairs from flight dynamics models into input and teacher signals
3. Perform supervised learning until output signal reaches acceptable delta with teacher signal
4. Validate model against PID controller

DEVELOP FLIGHT CONTROL WORKBENCH FOR MODEL FITNESS VERIFICATION

A workbench would be required for training and validating the neural network during offline and online learning. The neural network model should be able to be modified through this workbench. Training can be operated and monitored from this interface. This tool would also provide a graphical user interface (GUI) that dynamically plots the output signal against the teacher. It shall allow the capability of verifying and validating the model in its current state against a particular dataset. The plot lines in the graph will be two dimensional and allow displaying of the roll, pitch, and yaw values at each plot point. The values will be shown in a tabular view on the bottom of the screen, below the graph.

1. Develop workbench software for controlling, monitoring, and validating neural network training
2. Capability to create and modify neural network models
3. Capability to monitor training using two dimensional graph
4. Provide tabular view with context data of input, output, and teacher signals

TRAIN NEURAL NETWORK WITH SIMULATED AND FLIGHT TEST DATA

With the control system workbench in place, the neural network model can be trained against the flight dynamics model for use in the Twin-Engine aircraft. Training will occur until the output signal can reach a delta of 0.01 parity with the teacher signal. Data is retrieved from flight tests of the TwinEngine aircraft, as well as simulation in FlightGear. The tests include pitch and roll doublets. Noise is added to the closed loop simulation through a disturbance model. Both closed and open loop simulations are modeled in Simulink. Both MATLAB and C++ versions were used in offline training, while only the C++ version was used in online training.

TEST FITTED NEURAL NETWORK ON FLIGHT HARDWARE

Once the neural network is fitted, it will be capable of being integrated into the platform's flight computer. Providing PID turning to the PID feedback loop within the flight control system. This will allow proper hardware-in-the-loop testing to occur. The C++ version will run on the Epiphany processor of the Paralela board. Each neuron will run on a single thread per processor core. The flight computer communicates with the simulation on the ground segment over gigabit ethernet. Roll, pitch, and yaw data is wrapped in user datagram protocol (UDP) packets.

COMPARE RESULTS OF ECHO STATE NETWORK WITH MULTI-LAYER PERCEPTRON

Once software and hardware in-the-loop testing has occurred, the results of the recurrent model can then be compared to that of the multi-layer perceptron. The criteria of comparison will be the following.

1. Flight Data vs. Neural Network Response
2. Flight Data vs. Neural Network Model Response (Roll Rate)
3. Aircraft Response to Roll Rate Command
4. Response of the Aircraft in the Presence of Noise

During online training, a dual network model will be employed. The core model is the echo state network, with a multi layer preception used for error correction. This will occur in the PID loop. The dynamics signal is fed into the ESN with the error corrected signal being fed back into it. Online and offline training is compared to determine the differences between using a pure ESN model and a hybrid ESN/MLP hybrid. This will hopefully lead to future work in dual network architectures.

4. METHODOLOGY

This research effort looked into two classes of neural networks for accomplishing the objective of adaptive flight control: system identification and control. For system identification and control, offline training data was generated using the validated nonlinear flight dynamics models for the Twin-Engine aircraft. The dynamics model was developed with collected flight data from flight tests of the Twin-Engine.

The model was developed using Athena Vortex Lattice; a aerodynamic and flight-dynamic analysis program for modeling rigid aircraft surfaces and dynamics. It is derived using the nonlinear equations of motion that describe the input-output relationships for the aircraft. The model is loaded into an open source flight simulator, called FlightGear. This software allows for simulating a flight with the Twin-Engine, providing the ability to record the roll, pitch, and yaw of the aircraft throughout the duration.

These values were then used to train the echo state-based neural network and multi-layer perception during supervised offline training, as the teacher signal. The input signal is generated using the originating flight data to provide a starting point for the model.

Training will occur until the output signal from the network model is within an acceptable delta with the teacher signal.

4.1 SYSTEM IDENTIFICATION

A nonlinear flight dynamics model for the twin-engine airplane was developed using Athena Vortex Lattice (AVL) software as well as the System Identification Program for Aircraft (SIDPAC) software. SIDPAC software uses regression and maximum likelihood methods for the determinations of model parameters. The identified model was verified using flight data. The airplane was flown for doublet inputs in the aileron, elevator, and rudder. It is seen that the model response matches well with the flight data. There are some discrepancies. Further refinement of the flight dynamics model is underway. Using the validated flight dynamics model, a model for the airplane was created in the FlightGear Flight Simulator as shown in the preceding figure. The data required for the neural network training was generated using the FlightGear model of the airplane. Both

Figure 4.1: Twin-Engine Flight Dynamics Model

offline and online training was performed. The simulated data instead of real flight data was used because of the fact that the network training requires a large set of data for effective and accurate training. The generated data is separated into two categories, training and validation. The training data was used to generate the input signal, while the validation data was used to generate the teacher signal. These signals include but not limited to roll rate/acceleration, aileron deflection, sideslip angle, and yaw rate.

4.2 NEURAL NETWORK

Two different models, MATLAB/Simulink and FlightGear, were developed to acquire the data needed to train the neural networks. These models are based on a six-degree-of-freedom (6-DOF) nonlinear flight dynamics model of the airplane [15]. The data is used to first train the offline neural network and then used to initialize the online neural network. The online neural network is used to adapt to different flight conditions as well as reduced the inversion error. These two neural networks are implemented using C on FlightGear for software-in-the-loop(SIL) simulation. The online neural network is also used to eliminate the effect of disturbance and noise.

4.2.1 MODEL

The development of the echo-state based neural network was a two-phase process. This included an initial prototype and a production implementation. The model contains 17 neurons inside of the reservoir, a nonlinear hidden layer interconnected with the input and

output signals. The number of neurons was chosen based on the requirement for have one neuron per thread within the flight computer. As this would provide the best performance in-flight, in terms of concurrency and memory performance, from past experience. The input and output signals consists of vectors with 5 values each. Each value being a tuple of roll, pitch, and yaw values.

Weight matrices are defined for describing the signal strength between neurons and signals.

1. The input weight matrix is 5×17 and defines the connections between the linear input signal and the nonlinear reservoir.
2. The reservoir weight matrix is 17×17 and defines the connection between the neurons in the nonlinear reservoir.
3. The output weight matrix is 17×5 and defines the connections between the linear output signal and the nonlinear reservoir.

Neurons have 10 percent connectivity, providing a high degree is sparsity. This allows the model to stay somewhat simplified compared to other recurrent models. The reservoir exhibits a 40 percent leak rate. This provides a means for the current knowledge to leak out over time, without fill up non-volatile memory. Weight adjustment occurs through Weiner-Hopf based linear regression.

OFFLINE TRAINING

Offline training utilizes the data generated from the FlightGear model of the aircraft. The Training continues until the output signal comes with an acceptable delta with the teacher, which is set to be 0.01 percent. At this point, it is properly fitted and ready for the online training.

ONLINE TRAINING

For the online training process, the data used was the real-time data from the FlightGear simulator while the airplane was flying in simulation. The data was transmitted to the ESN

Figure 4.2: Offline Model

from the FlightGear simulator using the user datagram protocol (UDP). This was also useful in testing the controller performance in software-in-the-loop (SIL) simulation. The response of online trained ESN compared to the simulated flight data for roll rate in the following figure. The output of the ESN is fed into a Multi Layer Perceptron (MLP). The MLP is used for error correction, in order to reduce any unnecessary noise before passing along to the plant model. This It can be seen that there is an excellent correlation between the two. It can also be seen that the error decreases over time. The following figure shows the response of offline and online trained ESN compared to the simulated flight data for aileron. It can be seen that there is excellent correlation between the simulated flight data and the online trained model response. With the online training, the error between the response of the offline trained model and the simulated data has largely disappeared. For online training, a dual network approach will be used. This modeling technique has gained a lot of traction in recent years. It entails a main model used for training and a secondary model used for error correction. In this case, an Echo State Network is used as the core model, with a Multi Layer Perception used as the error corrector. This allows for reducing noise of the signal as it is fed into the plant of the control system. Noise can be generated from varying environmental conditions, such a wind gusts. This will be prevalent in the results section, as wind gusts are used to provide variation in the plots provided. A comparison between pure and hybrid ESNs will be conducted.

4.2.2 MATLAB PROTOTYPE

The prototype was modeled in the MATLAB within the Simulink environment. Both offline and online training was performed. The offline training realizes the inversion required for the feedback linearization. The online training is then performed to reduce the modeling errors. Both training methods use Weiner-Hopf method for linear regression. This allows for the generation of the output weights, which is then used to generate the output signal. The output signal is fed back into the network as a means of learning. The feedback loop eliminates the requirement of backpropagation of weights, as the output signal is used in calculating the next current state.

4.2.3 C++ IMPLEMENTATION

The C++ implementation supports Unix-based and Windows operating systems. It has been integrated into the avionics system as an application running on a QNX Neutrino partition. The matrix algebra is handled by the Eigen2.0 C++ library. The extensibility of this library allows for a well-defined hierarchy of the matrix- and vector-based behavior of signals and weight matrices in the network. Both offline and online trainings were performed. The offline training realizes the inversion required for the feedback linearization. The online training is then performed to reduce the modeling errors. Both training methods use Weiner-Hopf method for linear regression. This allows for the generation of the output weights, which is then used to generate the output signal. The output signal is fed back into the network as a means of learning. The feedback loop eliminates the requirement of backpropagation of weights, as the output signal is used in calculating the next current state.

The C++ implementation came into fruition after verification and validation of the prototype had completed. With the model being sound and complete, a production implementation was required for deploying on the aircraft. This was implemented with 17 neurons and a 40 percent leak rate. Weight matrices for neuron connections are handled by the Eigen C++ linear algebra matrix. The framework needed to be modified in order to support pseudoinverse computation, which is used in generating the output signal using

Figure 4.3: Hyperbolic Tangent

Figure 4.4: Closed Loop

the Weiner-Hopf method. The target signal is generated from data extracted from the Twin Engine flight dynamics model (FDM) within FlightGear. The C++ implementation has been trained using offline and online training.

4.2.4 CLOSED-LOOP SYSTEM

The following figure shows the implementation of the trained networks in a closed-loop. The figure also shows the proportional-integral-derivative (PID) controllers in the loop. The PID controller is used to shape the closed-loop response as well as to guarantee the stability. In this figure, the signal $u[pd]$ is the output from the PID block, and $u[ad]$ is the adaptive term that is realized by the online network. The figure shows the closed-loop response of the airplane to a unit step command in bank angle with the offline only and offline and online trained models in the loop. It is seen that the closed-loop response is

Figure 4.5: FlightGear Simulation

superior with both the models in the loop compared to the response with only the offline network.

4.3 SIMULATION

The Software-in-the-Loop simulation is built on top of FlightGear, an open source flight simulation program. The simulation is written in C++, with the graphics rendering in OpenGL. FlightGear provides a simulation experience that is realistic used in academia mostly. The tool is used by NASA, MathWorks, Arizona State, University of Illinois at Urbana Champaign, and Pennsylvania State University.

As mentioned previously, the flight dynamics model for the Twin-Engine aircraft was developed using JSBSim, an open source flight dynamics modeler. This was taken and turned into a FlightGear aircraft package. The flight dynamics model package is driven by an XML file that defines metrics (Wing span, Wing area, etc.), mass balance, propulsion, etc.

In order to create an aircraft model within FlightGear, stability and control derivatives, moment of inertia, and 3-D geometric model are required. The stability and control derivatives were calculated using AVL, which were verified using system identification. The moments of inertia for the aircraft were experimentally calculated using a bifilar

method. This method involves swinging the aircraft using a bifilar about each axis. During the swinging for the aircraft acceleration and velocity data are recorded along with the period of oscillation. This data is then used to estimate the moments of inertia using parameter identification in MATLAB/Simulink. The 3-D geometric model was created using SolidWorks. This model was then converted to an acceptable file format for FlightGear.

A C++ driver was written to interface between the flight simulation and the neural network controller. The C++ controller communicates with FlightGear's Generic Protocol. This communication interface utilizes the Usergram Data Protocol (UDP). UDP packets are sent to and received from FlightGear which contain the flight parameters of the simulation. The property list in the packets are defined in XML form. An example is given below.

```
<?xml version="1.0"?>
<PropertyList>
  <generic>

    <output>
      <line_separator>newline </line_separator>
      <var_separator>newline </var_separator>
      <binary_mode>>false </binary_mode>

      <chunk>
        <name>speed </name>
        <format>V=%d </format>
        <node>/ velocities / airspeed -kt </node>
      </chunk>

      <chunk>
        <name>heading (rad) </name>
        <format>H=%.6f </format>
        <type>float </type>
```

```

        <node>/orientation/heading-deg</node>
        <!-- degrees to radians -->
        <factor>0.0174532925199433</factor>
    </chunk>

    <chunk>
        <name>pitch angle (deg)</name>
        <format>P=%03.2f</format>
        <node>/orientation/pitch-deg</node>
    </chunk>
</output>

</generic>
</PropertyList>

```

The packets are received in binary form, in a series of 4 byte floating point and decimal integers. The C++ driver receives and disseminates the flight parameters in the binary package. The flight dynamics (roll, pitch, and yaw) are sent to the neural network controller for tuning. The dynamics parameters are sent back to the C++ driver, which packages it into a binary packet and sent back to FlightGear over UDP. This is used to update the roll, pitch, and yaw of the simulated Twin-Engine.

The data required for echo state network training may be obtained in different ways. Flight data collected during flight tests can be used for training. Alternatively, the data needed to train the neural network can be obtained using flight validated simulation models. There are two ways to generate simulation data: one is from the nonlinear flight dynamics model using Simulink and second is using the FlightGear model of the airplane. To get data from the nonlinear model a sinusoidal input for the aileron deflection.

```

ARMED
MANUAL> Mode MANUAL
GPS lock at 582 meters
Received 462 parameters
wp list
MANUAL> Requesting 8 waypoints t=Sat May 30 05:34:45 2015 now=Sat May 30 05:34:4
5 2015
16 0 -35.3632621765 149.1652374268 584.380005 p1=0.0 p2=0.0 p3=0.0 p4=0.0 cur=0
auto=1
22 3 -35.3611640930 149.1639862061 28.110001 p1=15.0 p2=0.0 p3=0.0 p4=0.0 cur=0
auto=1
16 3 -35.3594665527 149.1616973877 99.800003 p1=0.0 p2=0.0 p3=0.0 p4=0.0 cur=0 a
uto=1
16 3 -35.3663330078 149.1626586914 100.730003 p1=0.0 p2=0.0 p3=0.0 p4=0.0 cur=0
auto=1
16 3 -35.3661308289 149.1645812988 100.000000 p1=0.0 p2=0.0 p3=0.0 p4=0.0 cur=0
auto=1
16 3 -35.3592720032 149.1637573242 100.000000 p1=0.0 p2=0.0 p3=0.0 p4=0.0 cur=0
auto=1
177 0 0.0000000000 0.0000000000 0.000000 p1=2.0 p2=-1.0 p3=0.0 p4=0.0 cur=0 auto
=1
16 3 -35.3592720032 149.1637573242 100.000000 p1=0.0 p2=0.0 p3=0.0 p4=0.0 cur=0
auto=1
auto

```

Figure 4.6: FlightGear Generic Protocol

4.4 PLATFORM

At the heart of the avionics platform is the Parallela board, which acts as the vehicle management computer (VMC). It consists of a 16 core Epiphany processor and a Zync chip, which includes an FPGA and dual core A9. Providing 18 processing cores in total. The Parallela sports 1GB of SDRAM and a Micro-SD slot. It can interface with other flight components through a 48-pin GPIO and gigabit ethernet adapter. It has a power requirement of 5W.

The neural network runs within a PID control loop. The PID controller runs on a single thread, considered the master. Each neuron within the network runs on its own thread. The results of the neuron activations are collected by the master thread and computed into an output signal. This is compared with the teacher signal to trigger weight adjustments of the neurons. The fitted output signal is fed into the PID control loop, providing the roll, pitch, and yaw values to tune the stability of the aircraft.

4.5 AIRCRAFT

The aircraft tested against the model being developed is the Twin-Engine aircraft. The DA 50 gasoline engine-powered high-wing twin-engine airplane has wing span of 134 inches

Figure 4.7: Parallela Board

and is 95 inches long. The empty weight of the airplane is 42 lbs. with 25 lbs. of payload capacity.

4.5.1 FLIGHT DYNAMICS MODEL

Two different models, MATLAB/Simulink and FlightGear, were developed to acquire the data needed to train the neural networks. These models are based on a six-degree-of-freedom (6-DOF) nonlinear flight dynamics model of the airplane [15]. The data is used to first train the offline neural network and then used to initialize the online neural network. The online neural network is used to adapt to different flight conditions as well as reduced the inversion error. These two neural networks are implemented using C++ on FlightGear for software-in-the-loop(SIL) simulation. The online neural network is also used to eliminate the effect of disturbance and noise.

The dynamics model of an aircraft is the representation of the flight dynamics of the aircraft. The model predicts the behavior of the aircraft following control inputs. The model is developed from equations of motion that describe the rigid body motion of the aircraft. There are three force and three moment equations that describe this motion as given below.

$$\mathbf{U} = RV + WQ - g \sin \theta + \frac{F_x}{m} \quad (4.1)$$

$$\mathbf{V} = -UR + WP + g \sin \phi \cos \theta + \frac{F_y}{m} \quad (4.2)$$

$$\mathbf{W} = UQ - VP + g \cos \phi \cos \theta + \frac{F_z}{m} \quad (4.3)$$

$$\mathbf{P} = (c_1R + C_2P)Q + c_3L + c_4N \quad (4.4)$$

$$\mathbf{Q} = c_5PR + c_6(P^2 - R^2) + c_7M \quad (4.5)$$

$$\mathbf{R} = (c_8P - c_2R)Q + c_4L + c_9N \quad (4.6)$$

The total force and moments result from contributions of aerodynamic, propulsive, and gravitational forces. The forces and moments are functions of instantaneous values of perturbation variables. Perturbation variables are the instantaneous changes from the reference conditions. To completely describe the aircraft motion, three kinematic equations given below are required.

$$\phi = P \tan \theta (Q \sin \phi + R \cos \phi) \quad (4.7)$$

$$\theta = Q \cos \phi + R \sin \phi \quad (4.8)$$

$$\psi = \frac{Q \sin \phi + R \cos \phi}{\cos \theta} \quad (4.9)$$

The equations given above are nonlinear. However, these equations are not suitable for the design of control systems for the vehicles. Paragraphs below describe the nonlinear and linear flight dynamics models that can be used for control system design.

Figure 4.8: Twin Engine Aircraft

Taking in to account the lift, drag, gravitational, and thrust forces and writing these equations in coefficient form, the equations above can be written in the following form. Using these equations, a nonlinear simulation model was. The model response will be compared with flight data for validation. The validated model will then be used for generating data required for neural network training. Figure 2 shows the nonlinear model being developed in Simulink. The stability and control derivatives for the Twin-Engine aircraft were calculated using AVL and by also using system identification.

4.6 FLIGHT LAB

The Flight Lab workbench is software developed as part of this research effort to provide a tool to develop and maintain flight control techniques. This includes but is not limited to neural network, radial basis function, and nonlinear dynamic inversion models. The tool was written in C++. The graphical user interface was developed with Qt. The tool allows for loading and training control models. Control models can be saved into JSON (JavaScript Object Notation) format. The training process is mapped out through a 2D-plotted map. The map consists of the plots for the output and teacher signals over time. Selecting a particular plot displays a context pop-out showing the roll, pitch, and yaw values at each time step.

Training can be executed by menu or console. The process can be paused and saved for future training at a later date. The current state of the network model is persisted into the JSON file. In addition, the output of the network can be retrieved in-flight. This is

Figure 4.9: Flight Lab

accomplished with use of a UDP connection. It requires a wireless adapter on the computer running this tool, to be connected to the Bullet WiFi adapter running on the Twin-Engine aircraft.

5. EVALUATION OF RESULTS

This figure shows the response of online trained ESN compared to the simulated flight data for roll rate. It can be seen that there is an excellent correlation between the two. It can also be seen that the error decreases over time.

Figure 5.5 shows the response of offline and online trained ESN compared to the simulated flight data for aileron. It can be seen that there is excellent correlation between the simulated flight data and the online trained model response. With the online training, the error between the response of the offline trained model and the simulated data has largely disappeared.

The paper showed that a recurrent neural network can successfully be used for the identification and nonlinear control of flight vehicles. Echo state network was used for both offline and online training using the input/output pairs for the twin-engine UAV. The offline training realizes the inversion required for the feedback linearization. The data required for the offline training was generated using the FlightGear model of the aircraft that is based on a validated nonlinear dynamics model of the aircraft. The online training is then performed to reduce the inversion errors. Simulation results for roll rate and bank angle show that with the inclusion of online trained network, the performance of the controller significantly improves for both the open-loop and closed-loop responses. Because of the recurrent behavior, the ESN facilitates adaptability with more ease than a feedforward network, and provides shorter bursts while staying stable. It was also found that the error due to offline training was less compared to other networks. Future work will involve including the controllers for pitch and yaw-axes and determining candidate Lyapunov function in order to prove that the controllers are globally asymptotically stable. Future work will also involve testing the ESN controllers in SIL and Hardware-in-the-Loop (HIL) simulations. If found satisfactory in these simulations, the controller will be tested in flight. The data required for the online training will be the actual flight data. Also, the controllers will include adaptive elements for increased robustness in the presence of modeling errors, disturbance, and noise.

This research was supported by National Science Foundation under the Award No.

Figure 5.1: Twin Engine Aileron Input Command

Figure 5.2: Plant Input and Output

1102382. The authors would like to thank Hovig Yaralian, Luis Andrade, and Nigam Patel for their help with flight testing, data collection, and development and validation of the nonlinear flight dynamics model of the UAV.

The design for the offline neural network began with generating data from FlightGear model of the Twin-Engine UAV. Two sets of data were collected: one for the training of the neural network and the second to validate the network. To train the network the aileron deflection angle, beta, and roll rate were recorded from FlightGear. To calculate the roll acceleration, roll rate was numerically differentiated in MATLAB. The inputs to the neural network consisted of roll acceleration, roll rate, and beta. The output of the network was the aileron deflection. The figure titled Plant Input and Output shows the inputs and the desired output for the offline neural network controller.

The training for the offline network begins by first normalizing the aileron deflection between 0 and 1 so the Log-Sigmoid function can be applied. The aileron deflection is normalized because the output of the Log-Sigmoid function will be between 0 and 1. If the deflection was not normalized there would be errors when calculating the derivative of the output layer. Once the data has been normalized random weights are initialized. These weights are then used along with the inputs in the forward phase of the neural network. The forward phase will provide an estimated output which is compared with the desired output to calculate the error. This error is then used in the backward phase to update the weights of the network.

The learning rate for the offline network is chosen to be .01. The forward and backward phases are performed for each training data sample. The training runs for each training sample in the training set for a set number of epochs. The epoch for the offline neural network is set to be 1000. Once the training is complete the neural network is validated using the validation data. To validate the network a forward phase is performed using the trained weights and the validation flight data.

As can be seen in these figures, the model response matches well with the flight data. A further improvement in the model is required to increase the accuracy of the model. More flight data needs to be collected for better flight dynamics model using techniques such as system identification techniques.

Figure 5.3: Commanded Bank Angle

Figure 5.4: Online Response with Gusts

The error seen in the figures is caused by the error in numerical differentiation and dynamic inversion mapping. This error can be reduced when using online training which is discussed later on in the chapter.

Once the offline neural network has been trained a neural network based controller is designed using custom C++ code.

To test the performance of the offline and online controllers a simple maneuver was created. The maneuver consisted of a step command 5 seconds into the flight and a following step command was around 25 seconds into the flight. The first step command was for bank angle to be 10 degrees and the following command was for 5 degrees. The figure titled Commanded Bank Angle shows the commanded bank angle used to test the two controllers.

The linear compensator used in the above simulation was a Proportion-Derivative (PD)

controller. The values for k_p and k_d were not changes between the online and offline controller. The values for k_p and k_d were 10 and 3 respectively. Both controllers perform the same when there are no disturbances. The offline controller does have some oscillation due to the error in model the aircraft dynamics. This oscillation is eliminated when using the online controller because the online controller adapts to the error in the model dynamics. To test the performances of the controller in the presence of disturbances, gust is added into FlightGear using the environmental properties. The strength of the gust is set to 25 percent.

As seen from the figure the online controller works well in the presence of Gust. This online controller can also work in the presence of component failure and sensor noise. The adaptation law incorporated in the online neural network training can minimize any type of error.

The adaptive weightings for the system are updated every time step to compensate for the errors associated with the presence of gust. The weighting for the system is updated according to the adaptive laws. A quick change is seen in the weighting for the initial part of the maneuver to correct for the undesired roll rate in the opposite direction. Then the weighting levels off to a steady state value once the system has reached the commanded state. The figure titled Online Response with Gusts shows the results of the online controller and the offline controller in the presence of noise.

As seen from the above figure the online controller performs better when noise is added. The noise used for the simulation is white Gaussian noise. The noise is added to the feedback loop. The offline controller seems to follow the command however it has a lot of oscillation. The oscillation trend is similar to the noise added in the loop. The online controller adapts to the noise and has little oscillation due to the noise added. Along with comparing the offline and online controller a PID controller is compared to the online controller.

As seen in the figure above the PID controller is unstable and unable to control the aircraft in the presence of noise. This is due to the limitation of a PID controller, which cannot adapt to disturbances or noise. For a controller to adapt to noise and disturbance the controller needs to have adaptation incorporated into it. PID controller is used alongside

Figure 5.5: Response Comparison of Online and Offline Controller

Figure 5.6: Inputs and Desired Output to Offline Neural Network

an online controller to shape the response of the controller.

The flight dynamics (roll, pitch, and yaw) data was tuned in the PID control loop within the Simulink model utilized during software-in-the-loop simulation. Amplitude stemming from bank angle and yaw rate were used to derive the aileron signal. A series of training epochs were conducted to find a target signal that was properly fitted against the aileron generated. It was found that the generated aileron was had a 0.01 percent delta with the desired target. It took three training epochs to validate the model. This allowed the confident needed to move on to training with simulated roll rates.

The simulated roll rate was obtained from the control system plant of the Simulink model. The aileron starts at zero degrees and oscillates between 1 and -1 degrees. Once again, the PID control look within the Simulink model was utilized during software-in-the-loop simulation. It took a single training epoch to validate the model. There was small bursting

Figure 5.7: Simulated Roll Rate

seen as the aileron approached the local minimum and maximum of -1 and 1. A delta of 0.08 percent was found. The bursting was overcome by training subsets of the signal with small variations between the local minimum and maximum values. This was able to be accomplished using the secondary Multi Layer Perceptron (MLP) as an error correction mechanism, during online training. The results provided the validation to show that the dual neural network architecture was a positive decision. In the future, it might be fruitful to try supplementing the MLP with a fuzzy logic controller. This has been gaining much popularity in academia.

As evaluation of the neural network continued, an aileron input command was created. With this command, the local minimum and maximum now ranged between 30 and -30 degrees. It dampens to 0 degrees at around 12 seconds. This will now be used to simulate the Twin Engine aircraft's roll rate output. This can then be used to generate roll doublets during simulated flight tests with flight gear. The data aid in demonstrating maneuverability across the six degrees of freedom of the aircraft in varying environment conditions. Such conditions include wind gusts. This will be shown in a later figure. The roll rate output was generated and used during offline training of the echo state network. It took four training epochs, with a validated model being reached. An error delta of 0.02 percent was seen. This provided a fitted model for our simulation.

With the trained model, roll rate output from the ESN was able to be obtained. This can be seen in Figure 5.10. As was expected, there was bursting and variation between the teacher and neural activation signal at the local minimum and maximum of 250 and -250

Figure 5.8: Aileron Input Command

Figure 5.9: Roll Rate Output

Figure 5.10: ESN Roll Rate

degrees respectively. As the roll rate dampened around 0, the variation was almost non-existent. More training will be required and the peaks and valleys of the signal. It was seen during online training, that error correction was able to overcome some variation using the Multi Layer Perceptron neural network model.

The pitch rate was generated using the FlightGear simulation. It includes delta and q, that are degree and degree per second respectively. The echo state network was then fitted against the simulation data. Very little bursting was seen, with a low error term.

The ESN online model response shows the training over several epochs. To test the performance of the offline and online control systems, a simplistic maneuver was used.

This included a step command at 5 seconds in. Another step command at 25 seconds. The first command takes the form of a bank angle. Figure 5.14 shows the training results. The linear compensator used in the simulation was a Proportion-Derivative (PD) controller.

The values for k were not changed between online and offline training. There did not seem to be much noise during training. The oscillation seen offline, was corrected online with the Multi Layer Perceptron.

Wind gusts were used to further test the oscillations. This was added using the FlightGear simulation. The gust was at about 25 percent. This can be seen on Figure 5.16. Online training seemed to fair better with handing gusts.

Figure 5.11: Pitch Rate

Figure 5.12: ESN Online Model Response

Figure 5.13: Commanded Bank Angle

Figure 5.14: Closed Loop Response for Bank Angle

Figure 5.15: ESN Bank Angle Response

Figure 5.17 shows the PID tuner with ESN and MLP neural network models. The added noise threw off the model, as the fundamental logic doesn't support adaptation. The Echo State Network aided in this. In the future, the ESN will be developed further to better support adaptation. The C++ and MATLAB code can be found in chapter 8.

Figure 5.16: Online Response with Gusts

Figure 5.17: PID Tuning with ESN and MLP

6. CONCLUSION AND FUTURE WORK

This thesis showed that a recurrent neural network can successfully be used for the identification and nonlinear control of flight vehicles. The echo state network was used for both offline and online training using the input/output pairs for the twin-engine UAV. The offline training realizes the inversion required for the feedback linearization. The data required for the offline training was generated using the FlightGear model of the aircraft that is based on a validated nonlinear dynamics model of the aircraft. The online training is then performed to reduce the inversion errors. Simulation results for roll rate and bank angle show that with the inclusion of online trained network, the performance of the controller significantly improves for both the open-loop and closed-loop responses.

Because of the recurrent behavior, the ESN facilitates adaptability with more ease than a feedforward network, and provides shorter bursts while staying stable. It was also found that the error due to offline training was less compared to other networks.

Future work will involve including the controllers for pitch and yaw-axes and determining candidate Lyapunov function in order to prove that the controllers are globally asymptotically stable. Future work will also involve testing the ESN controllers in SIL and Hardware-in-the-Loop (HIL) simulations. If found satisfactory in these simulations, the controller will be tested in flight. The data required for the online training will be the actual flight data. Also, the controllers will include adaptive elements for increased robustness in the presence of modeling errors, disturbance, and noise.

7. GLOSSARY

<u>Symbol</u>	<u>Description</u>
A	System Matrix
Aileron	A hinged surface
AI	Artificial Intelligence
ANN	Artificial Neural Network
Aerial	Existing, happening, or operating in the air
AoA	Angle of Attack
b	Wing span
B	Input Matrix
C	Output Matrix
D	Input-Output Coupling Matrix
Elman	A three layer neural network
g	Gravity constant
Jordan	Class of neural network
Y	Learning rate
h	Geometric constant
L	Aerodynamic moment rate along x-axis
M	Aerodynamic moment rate along y-axis
Ma	Mach number
N	Aerodynamic moment rate along z-axis
P	Angular rate in x-axis
R	Angular rate in z-axis direction
S	Surface area
U	Velocity component in y-axis direction
r	Angular rate in z-axis direction
V	Velocity component in y-axis direction
W	Velocity component in z-axis direction

<u>Symbol</u>	<u>Description</u>
NAS	National Air Space
ESN	Echo State Network
MLP	Multi Layer Preception
RBF	Radial Basis Function
X	Input Vector
W	Reservior Weight Matrix
y	Output signal
D	Teacher signal
F	Hyperbolic tangent based neuron activation function
G	Logistic sigmoid based linear regression function
PID	Proportional-Integral-Derivative
FCS	Flight Control System
GUI	Graphical User Interface
AVL	Athena Vortex Lattice
SIDPAC	System Identification Program for Aircraft
FlightGear	Open Source Flight simulation software
UDP	User Datagram Protocol
SIL	Software in the Loop
HIL	Hardware in the Loop
MATLAB	Matrix Laboratory software
Weiner-Hopf	Linear regression method
C	High level programming language
FDM	Flight dynamics model
CLS	Closed Loop system
VMC	Vehicle Management Computer
MMC	Mission Management Computer

<u>Symbol</u>	<u>Description</u>
Cnb	Lateral Sability Coefficient
Cnp	Lateral Stability Coefficient
Cn	Lateral Stability Coefficient
Cy	Lateral Stability Coefficient
Dd	Input-Output Coupling Matrix

8. SOURCE CODE

```
/**
 * Flight Control Model
 */
int main()
{
    // Start esn model
    ESN *esn = new ESN();
}

Matrix::Matrix()
{
}

Matrix::Matrix(int x, int y)
{
}

/**
 * Echo State Network class for used in a fitted
 * flight control model.
 */

// Constructor
ESN::ESN()
{
}

clc
```

```

clear

% Read log file
Data = xlsread('Fri_May_24_14-54-11_2013e.xlsx');

% Read in time (ms)
t = Data(:,1)-27081;

% Convert milli seconds to seconds
tsec = t./1000;

% Convert seconds to minutes
tmin = tsec./60;

% Determine the point at which 1 Hz recording
% to 10 Hz recording
tdiff = diff(tsec);

%Change time set to only contain 10 Hz data
tsec2 = tsec(829:4557);

%Roll Rate
P = Data(:,32).*(180/pi());
Pflight = P(829:4557);

% Pitch Rate
Q = Data(:,33).*(180/pi());
Qflight = Q(829:4557);

% Yaw Rate

```

```

R = Data(:,34).*(180/pi());
Rflight = R(829:4557);

% Roll
Roll = Data(:,38).*(180/pi());
Roll-flight = Roll(829:4557);

% Pitch
Pitch = Data(:,39).*(180/pi());
Pitch-flight = Pitch(829:4557);

% Yaw
Yaw = Data(:,40).*(180/pi());
Yaw-flight = Yaw(829:4557);

% Acknowledgement Ratio
AckRatio = Data(:,49);
AckRatio-flight = AckRatio(829:4557);

% RSSI
RSSI = Data(:,50);
RSSI-flight = RSSI(829:4557);

% Surface 0
% Aileron
Surface0 = Data(:,51);
Sur-flight = Surface0(829:4557).*(180/pi());

% Looking for constant surface deflection
Aileron1-diff = diff(Sur0-flight);

```

```

% Surface 1
% Elevator
Surface1 = Data(:,52).*(180/pi());
Sur1-flight = Surface1(829:4557);

% Looking for constant surface deflection
Elev1-diff = diff(Sur1-flight);

% Surface 2
% Throttle
Surface2 = Data(:,53);
Sur2-flight = Surface2(829:4557);

% Looking for constant surface deflection
Thrott1-diff = diff(Sur2-flight);

% Surface 3
Surface3 = Data(:,54).*(180/pi());
Sur3-flight = Surface3(829:4557);

% Looking for constant surface deflection
Sur3-diff = diff(Sur3-flight);

% Surface 4
Surface4 = Data(:,55).*(180/pi());
Sur4-flight = Surface4(829:4557);

% Looking for constant surface deflection
Sur4-diff = diff(Sur4-flight);

```

```

% Surface 5
% Aileron
Surface5 = Data(:,56).*(180/pi());
Sur5-flight = Surface5(829:4557);

% Looking for constant surface deflection
Aileron2-diff = diff(Sur5-flight);

% Surface 6
% Elevator
Surface6 = Data(:,57).*(180/pi());
Sur6-flight = Surface6(829:4557);

% Looking for constant surface deflection
Elevator2-diff = diff(Sur6-flight);

% Surface 7
% Throttle
Surface7 = Data(:,58);
Sur7-flight = Surface7(829:4557);

% Looking for constant surface deflection
Throttle2-diff = diff(Sur7-flight);

% Roll 3-2-1-1
% 1567-1624 No Elevator Movement
subplot(3,2,1)
plot(tsec2(1567:1624), Sur0-flight(1567:1624))
xlabel('time_(seconds)')

```



```

ylabel('deflection_(degrees)')
title('Roll_3-2-1-1')

subplot(3,2,2)
plot(tsec2(1567:1624),Roll-flight(1567:1624))
title('Roll_Response')
xlabel('time_(seconds)')
ylabel('roll_angle_(degrees)')

% 1733-1805 No Elevator Movement
subplot(3,2,3)
plot(tsec2(1733:1805),Sur0-flight(1733:1805))
xlabel('time_(seconds)')
ylabel('deflection_(degrees)')

subplot(3,2,4)
plot(tsec2(1733:1805),Roll-flight(1733:1805))
xlabel('time_(seconds)')
ylabel('roll_angle_(degrees)')
% 2563-2628 No Elevator Movement

subplot(3,2,5)
plot(tsec2(2563:2628),Sur0-flight(2563:2628))
xlabel('time_(seconds)')
ylabel('deflection_(degrees)')

subplot(3,2,6)
plot(tsec2(2563:2628),Roll-flight(2563:2628))
xlabel('time_(seconds)')
ylabel('roll_angle_(degrees)')

```

```

% Pitch 3-2-1-1
% 908 - 980 No Aileron Movement
figure
subplot(3,2,1)
plot(tsec2(908:980),Sur1-flight(908:980))
xlabel('time_(seconds)')
ylabel('deflection_(degrees)')
title('Pitch_3-2-1-1')

subplot(3,2,2)
plot(tsec2(908:980),Pitch-flight(908:980))
title('Pitch_Response')
xlabel('time_(seconds)')
ylabel('roll_angle_(degrees)')

% 1068 - 1148 No Aileron Movement
subplot(3,2,3)
plot(tsec2(1068:1148),Sur1-flight(1068:1148))
xlabel('time_(seconds)')
ylabel('deflection_(degrees)')

subplot(3,2,4)
plot(tsec2(1068:1148),Pitch-flight(1068:1148))
xlabel('time_(seconds)')
ylabel('roll_angle_(degrees)')

% 2380 - 2437 No Aileron Movement
subplot(3,2,5)
plot(tsec2(2380:2437),Sur1-flight(2380:2437))

```

```

xlabel('time_(seconds)')
ylabel('deflection_(degrees)')

subplot(3,2,6)
plot(tsec2(2380:2437),Roll-flight(2380:2437))
xlabel('time_(seconds)')
ylabel('roll_angle_(degrees)')

figure (3),
subplot(2,1,1),
plot(tsec2(1567:1624),
Sur0-flight(1567:1624),
'k-',
'linewidth',
3)
grid
ylabel('Aileron_(deg)','fontsize',25)
title('Roll_Doublet','fontsize',25),set(gca,'fontsize',25)
subplot(2,1,2),
plot(tsec2(1567:1624),
Roll-flight(1567:1624),
'k-',
'linewidth',
3)
ylabel('Roll_Angle_(deg)','fontsize',25),
grid,set(gca,'fontsize',25)
xlabel('Time_(sec)')

% number of epoch periods (training iterations)
t = 10;

```

```

% current epoch period
i = 1;

% number of neurons
n = 5;

% number of input signals
k = 1;

% number of output signals
l = 1;

leak-rate = 0.4;

% percentage of connected neurons
sparsity = 0.10;

% get flight parameters
run('setup-flight-params');

% initialize flight data
flight-data = get-flight-data();

% generate teacher signal (nx1)
d = generate-teacher-signal(l,t);
%d(i) = flight-data(roll,i);
%d(i+1) = flight-data(roll,i);
%d(i+2) = flight-data(roll,i);

```

```

d(i) = 60;
d(i+1) = 60;
d(i+2) = 60;

% generate input signal
u = generate-input-signal(t,k);

% generate output signal
y = generate-output-signal(t,l);

% input weight matrix
w-in = rand(n,k);

% feedback weight matrix
w-fb = rand(n,l);

% reservior weight matrix
w = sprand(n, n, sparsity);

% setup reservior matrix
x = zeros(n,t);

%setup w-out
w-out = [];

for train=1:10 % run the training algorithm for 3 epochs
    x(:,i) = recompute-reservior(
        u, w-in, x, w, y, w-fb, i);

% harvest neurons into an extended system state matrix

```

```

z = [x(1:n,i);u(i)];

% create collection matrices
S = transpose(z); % fill z row-wise
D = transpose(d(i)); % fill d row-wise

% create correlation matrices
R = transpose(S)*S;
P = transpose(S)*D;

% generate output weight matrix (w-out) through
% linear regression
w-out = transpose(pinv(R)*P);

% read-out phase (generate output signals)
y(i) = w-out(1,1:(n+k))*z;

% transition to next epoch
i = i + 1;
end

% plotting

% plot roll
u-plot = [u(1,1), u(1,2), u(1,3)];

d-plot = [d(1), d(2), d(3), 60, 60, 60, 60, 60];
y-plot = [0, 56, 60, 60, 62, 61, 60];

```

```

%axis = [0,0,0];

%plot(u-plot , 'b—o');
plot(d-plot , 'b—o');
hold;
plot(y-plot , 'r—o');

%plotyy(d,axis,y,axis);

ylabel('Input_Signal_vs_Teacher_Signal_(Roll)',
'FontSize', 24);
legend('teacher_signal:_d(n)', 'output_signal:_y(n)',
'FontSize', 24);

function u = generate-input-signal(t,k)
u = zeros(k,t);
%u(k,1) = -0.0333;
%u(k,2) = -0.0333;
%u(k,3) = -0.0333;
u(k,1) = 0;
u(k,2) = 0.1;
u(k,3) = 0.2;
u(k,4) = 0.3;
u(k,5) = 0.4;
u(k,6) = 0.5;
u(k,7) = 0.6;
u(k,8) = 0.7;
u(k,9) = 0.8;
u(k,10) = 0.9;

```

```

function y = neural-network(x)
y = 1/(1+exp(x));
end

function new-x = recompute-reservior(u, w-in, x, w, y, w-fb, i)
    new-x = w-in*u(i);

    if (not(isempty(w-fb)))
        new-x = new-x + w-fb*y(i);
    end

    if (not(isempty(w)))
        new-x = new-x + w*x(:, i);
    end
end

flight-data = csvread('flight-data-pitch.csv');
plot(flight-data, 'k', 'LineWidth', 3);
hold;
plot(flight-data, 'k');

xlabel('Time_(s)', 'FontSize', 24);
ylabel('Pitch_Doublet', 'FontSize', 24);

ylabel('Input_Signal_vs_Teacher_Signal_(Roll)');
legend('teacher_signal:_d(n)', 'output_signal:_y(n)');

x = [0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5];
y = [57,57,57,57,57,57,57,57,57,57,57];

```



```
plot(x, y, 'k', 'LineWidth', 3);
```

```
a = [-0.5572, -0.7814; 0.7814, 0];
```

```
b = [1, -1; 0, 3];
```

```
c = [2, 7];
```

```
sys = ss(a, b, c, 0);
```

```
step(sys)
```

```
<?xml version="1.0"?>
```

```
<PropertyList>
```

```
<generic>
```

```
<output>
```

```
<line_separator>newline</line_separator>
```

```
<var_separator>tab</var_separator>
```

```
<chunk>
```

```
<node>/position/altitude-agl-ft</node>
```

```
<name>altitude to go</name>
```

```
<type>float</type>
```

```
<format>%03.2f</format>
```

```
</chunk>
```

```
<chunk>
```

```
<node>/orientation/heading-deg</node>
```

```
<name>Heading</name>
```

```
<type>float</type>
```

```
<format>%03.3f</format>
```

```
</chunk>
```

```
</output>
```

```
</generic>  
</PropertyList>
```

```
/* udpserver.c */
```

```
#include <sys/types.h>  
#include <sys/socket.h>  
#include <netinet/in.h>  
#include <arpa/inet.h>  
#include <netdb.h>  
#include <stdio.h>  
#include <unistd.h>  
#include <errno.h>  
#include <string.h>  
#include <stdlib.h>  
#include <math.h>
```

```
#define ROLL                0  
#define PITCH              1  
#define HEADING            2  
#define ROLL_RATE          3  
#define PITCH_RATE        4  
#define YAW_RATE           5  
#define X_ACCEL            6  
#define Y_ACCEL            7  
#define Z_ACCEL            8
```

```
int main(int argc, char* argv[])  
{
```

```

int sock_in , sock_out;
int addr_len , bytes_read;
char recv_data[1024];
char exitcode = 'N';
float fdmData[100];
float aileron_ctrl , pitch_ctrl;
struct sockaddr_in server_addr_in , client_addr_in;


        int msgL = 0;
        struct sockaddr_in server_addr_out;
        struct hostent *host_out;
        char send_data[1024];


// host config
if ((sock_in =
        socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
    perror("Socket");
    exit(1);
}
server_addr_in.sin_family = AF_INET;
server_addr_in.sin_port = htons(5000);
server_addr_in.sin_addr.s_addr = INADDR_ANY;
bzero(&(server_addr_in.sin_zero),8);
if (bind(sock_in, (struct sockaddr *)&server_addr_in ,
        sizeof(struct sockaddr)) == -1) {
    perror("Bind");
    exit(1);
}
addr_len = sizeof(struct sockaddr);

```

```

printf("\nUDPServer_Waiting_for_client_on_port_5000");
fflush(stdout);

// client config
host_out =
    (struct hostent *)gethostbyname((char *)"127.0.0.1");

if ((sock_out = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
{
    perror("socket");
    exit(1);
}

server_addr_out.sin_family = AF_INET;
server_addr_out.sin_port = htons(5010);
server_addr_out.sin_addr =
    *((struct in_addr *)host_out->h_addr);
bzero(&(server_addr_out.sin_zero),8);

while (1)
{
    // wait for new UDP packet, and read
    // it in to recv_data
    bytes_read =
        recvfrom(sock_in, recv_data, 1024, 0,
            (struct sockaddr *)&client_addr_in, &addr_len);
    recv_data[bytes_read] = '\0';
    // print where it got the UDP data
    // from and the raw data

```

```

printf("\n(%s, %d)_said:_",
       inet_ntoa(client_addr_in.sin_addr),
       ntohs(client_addr_in.sin_port));
printf("%s", recv_data);
fflush(stdout);

// Parse UDP data and store into float array
sscanf(recv_data, "%f%f%f%f%f%f%f%f",
&fdmData[ROLL],
&fdmData[PITCH],
&fdmData[HEADING],
&fdmData[ROLL_RATE],
&fdmData[PITCH_RATE], &fdmData[YAW_RATE],
&fdmData[X_ACCEL],
&fdmData[Y_ACCEL],
&fdmData[Z_ACCEL]);

// Print data to screen for monitoring purposes
printf("\nSensor_Data\n
      %f\t%f\t%f\n%f\t%f\t%f\n%f\t%f\t%f\n",
      fdmData[ROLL], fdmData[PITCH],
      fdmData[HEADING],
      fdmData[ROLL_RATE], fdmData[PITCH_RATE], fdmData[YAW_RATE],
      fdmData[X_ACCEL], fdmData[Y_ACCEL], fdmData[Z_ACCEL]);

// Proportional control for roll and pitch
aileron_ctrl = -0.05*(fdmData[ROLL] - 0);
pitch_ctrl = 0.1*(fdmData[PITCH] - 5);

// Limit elevator control inputs

```

```

if (fabs(pitch_ctrl) > 0.6) {
    pitch_ctrl = (pitch_ctrl/fabs(pitch_ctrl))*0.6;
}
// Limit aileron control inputs
if (fabs(aileron_ctrl) > 0.6) {
    aileron_ctrl = (aileron_ctrl/fabs(aileron_ctrl))*0.6;
}

// Construct a packet to send over UDP with control inputs
msgL = sprintf(send_data ,
    "%f\t%f\n", aileron_ctrl , pitch_ctrl);
// Print Control inputs to stdout
printf("\nControl_Signal\n%s\n",
    send_data);
// Send UDP packet
sendto(sock_out , send_data , strlen(send_data) , 0,
    (struct sockaddr *)&server_addr_out ,
    sizeof(struct sockaddr));

}
return 0;
}

```

```

filename = 'C:\Data.csv';
delimiter = ',';
startRow = 545;
formatSpec = '%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f %[^\\n\\r]';
fileID = fopen(filename , 'r');
dataArray = textscan(fileID , formatSpec , 'Delimiter' , delimiter ,
'HeaderLines' , startRow-1 , 'ReturnOnError' , false);

```

```

fclose(fileID);
Data = [dataArray{1:end-1}];
Data = unique(Data,'rows','stable');
clearvars filename delimiter startRow formatSpec
fileID dataArray ans;
Time = Data(:,1)-Data(1,1);
Rudder = Data(:,2);
Aileron = Data(:,3);
Elevator = Data(:,4);
Roll = Data(:,5); %deg
Pitch = Data(:,6); %deg
Heading = Data(:,7); %deg
P = Data(:,8); %deg/sec
Q = Data(:,9); %deg/sec
R = Data(:,10); %deg/sec
Beta = Data(:,11); %deg
Alpha = Data(:,12); %deg
V = Data(:,13); % kt

% [P_dot,Q_dot,R_dot] = P_Q_R_dot(P,Q,R,Time);

plot(Time, Roll, 'k-', 'linewidth', 3), ...
ylabel('Roll_Rate_(rad)',
'fontsize', 15), grid, set(gca, 'fontsize', 15)

save('TwinNN1_Gust');

data = load('Twin_Gust1.mat');
x = [data.P_dot'; data.Beta'; data.P'];
y_ = data.Aileron'; % Aileron (rad)

```

```

time = data.Time;
y_max = max(y);
y_min = min(y);
norm_y = (y-y_min)/(y_max-y_min);

data_test = load('Twin_Gust2.mat');
xT= [data_test.P_dot';data_test.Beta';
data_test.P'];
yT=_data_test.Aileron';
time_test = data_test.Time;

% bias_size = size(x,2);
% bias = ones(1,bias_size);
% x = [x;bias];
[N_I,Is] = size(x);
[N_O,Io] = size(y);
N_HL = 10;
L_R = .01;

W_I_HL = 2*rand(N_I,N_HL)-ones(N_I,N_HL);
W_HL_O = 2*rand(N_HL,N_O)-ones(N_HL,N_O);

for i = 1:100
    for j=1:Is
        patnum = j;
        a_HL = W_I_HL'*x(:,patnum);
        y_HL=_logsig(a_HL);
        a_out=_W_HL_O'*y_HL;
        y_out = logsig(a_out);
    end
end

```



```

        delta_O =
            (norm_y(:, patnum)-y_out)*((1-y_out)*y_out);
        C_W_HL_O = ((L_R*delta_O*y_HL))';
        W_HL_O=W_HL_O+C_W_HL_O';
        delta_HL =
            ((1-y_HL).*y_HL).*(W_HL_O*delta_O);
        C_W_I_HL = (L_R*delta_HL*x(:, patnum))';
        W_I_HL=W_I_HL+C_W_I_HL';

    end

end

% bias_size = size(xT,2);
% bias = ones(1,bias_size);
% xT = [xT;bias];
[N_I_T,IsT] = size(xT);
for k = 1:IsT
    a_HL = W_I_HL'*xT(:,k);
    y_HL=logsig(a_HL);
    a_out=W_HL_O'*y_HL;
    y_out(:,k) = logsig(a_out);
end

norm_yout = (y_out*(y_max-y_min))+y_min;
plot(time_test,norm_yout*.3,'black',
time_test,yT*.3,'black—','linewidth',3)
xlabel('Time_(sec)','fontsize',15)
ylabel('Aileron_(rad)','fontsize',15)
legend('Offline_Only','Flight_Data')
grid on

```

```

f = fopen( 'OfflineNN1.txt', 'w' );
fprintf( f, '%f\n%f\n%f\n', N_I, N_O, N_HL, y_max, y_min );
fprintf( f, '%f\n', W_I_HL );
fprintf( f, '%f\n', W_HL_O );
f = fopen( 'C:\OfflineNN1.txt', 'w' );
fprintf( f, '%f\n%f\n%f\n', N_I, N_O, N_HL, y_max, y_min );
fprintf( f, '%f\n', W_I_HL );
fprintf( f, '%f\n', W_HL_O );
f = fopen( 'C:\OfflineNN1.txt', 'w' );
fprintf( f, '%f\n%f\n%f\n', N_I, N_O, N_HL, y_max, y_min );
fprintf( f, '%f\n', W_I_HL );
fprintf( f, '%f\n', W_HL_O );

save( 'OfflineNN1.mat', 'W_I_HL',
      'W_HL_O', 'time_test', 'norm_yout' )

% figure(2)
% plot(time_test, xT)

data_NN1 = load( 'OfflineNN1.mat' );
da_NN1 = data_NN1.norm_yout; % output of NN1
W_I_HL_NN1 = data_NN1.W_I_HL;
W_HL_O_NN1 = data_NN1.W_HL_O;

data = load( 'Twin_Gust.mat' );
x = [ data.P_dot'; data.Beta'; data.P' ];
%_Xn=_wgn( size(x,1), size(x,2), .5);
%_x=_x+_Xn;

```

```

y_ = data.Aileron'-da_NN1; % Error in Aileron
time = data.Time;
y_max = max(y);
y_min = min(y);
norm_y = (y-y_min)/(y_max-y_min);

data_test = load('Twin_Gust2.mat');
xT = [data_test.P_dot';data_test.Beta';data_test.P'];
xT_ = xT_;
yT_ = data_test.Aileron'-da_NN1; % Test
time_test = data_test.Time;

% bias_size = size(x,2);
% bias = ones(1,bias_size);
% x = [x;bias];
[N_I,Is] = size(x);
[N_O,Io] = size(y);
N_HL = 10;
L_R = .01;

W_I_HL = 2*rand(N_I,N_HL)-ones(N_I,N_HL);
W_HL_O = 2*rand(N_HL,N_O)-ones(N_HL,N_O);
W_HL_O_plot = zeros(1000,10);

for i = 1:1000
    for j=1:Is
        patnum = j;
        a_HL = W_I_HL'*x(:,patnum);
        y_HL_ = logsig(a_HL);
    end
end

```

```

a_out_ = W_HL_O' * y_HL;
y_out = logsig(a_out);
delta_O =
(norm_y(:, patnum) - y_out) * ((1 - y_out) * y_out);
C_W_HL_O = ((L_R * delta_O * y_HL))';
W_HL_O_ = W_HL_O + C_W_HL_O';
delta_HL = ((1 - y_HL) .* y_HL) .* (W_HL_O * delta_O);
C_W_I_HL = (L_R * delta_HL * x(:, patnum))';
W_I_HL_ = W_I_HL + C_W_I_HL';
W_HL_O_plot(i, :) = W_HL_O';

end

end

% bias_size_ = size(xT, 2);
% bias_ = ones(1, bias_size);
% xT_ = [xT; bias];
[N_I_T, IsT]_ = size(xT);
for k_ = 1: IsT
a_HL_ = W_I_HL' * xT(:, k);
y_HL = logsig(a_HL);
a_out = W_HL_O' * y_HL;
y_out(:, k)_ = logsig(a_out);
end
norm_yout_ = (y_out * (y_max - y_min)) + y_min;
y_out_NN2_N11_ = norm_yout + da_NN1;
Aileron_ = data_test.Aileron';

for k = 1: IsT
a_HL = W_I_HL_NN1' * xT(:, k);

```

```

y_HL_ = logsig(a_HL);
a_out_ = W_HL_O_NN1'*y_HL;
y_out(:,k) = logsig(a_out);
end
norm_yout_NN1 = (y_out*(y_max-y_min))+y_min;

plot(time_test ,y_out_NN2_N11*.3,'black',time_test ,Aileron*.3,
'balckx',time_test ,norm_yout_NN1*.3,'black—', 'linewidth',3)
title('Aileron_Vs_Time','fontsize',15)
xlabel('Time_(sec)','fontsize',15)
ylabel('Aileron_(rad)','fontsize',15)
legend('Online_+_Offline','Flight_Data','Offline_Only')
grid on

f = fopen('C:\OfflineNN2.txt','w');
fprintf(f,'%f\n%f\n%f\n',N_I,N_O,N_HL,y_max,y_min);
fprintf(f,'%f\n',W_I_HL);
fprintf(f,'%f\n',W_HL_O);

```

d. UDP Communication

```

function [DataCatch] = CatchData(portcatch ,freq ,Time ,datafile)
    instrreset
    % Portcatch is the port the packets will be send to
    % Freq is the frequency
    % Time is the time to catch the data
    % File to save the data
    M=[];
    time = [];
    t = 0;
    % Create UDP

```

```

DataIN=udp( '127.0.0.1', 'LocalPort', portcatch );
% Open UDP
fopen( DataIN );
% Read Data
    while t < Time
        data=fscanf( DataIN, '%f_%f_%f' );
        disp( data )
        M = [M; data(1,1), data(2,1), data(3,1)];
        t = t + (1/freq);
        time = [time;t];
        set(gcf, 'color', 'white');
        drawnow;
        plot( time, M(:,2), 'k-', 'linewidth', 3), ylim([-1 1]), ...
            ylabel( '\delta', 'fontsize', 15),
            title( 'Defelction', 'fontsize', 15)
        grid on;
    end
%Close UDP
fclose( DataIN );
%Delete UDP
delete( DataIN );
%Save Data
save( datafile );

#include <iostream>
#include <fstream>
#include <Eigen>
#include <stdio.h>
#include <winsock2.h>
#include <string.h>

```

```

#include <stdlib.h>
#include <sstream>
#include <Windows.h>

# define SERVER "127.0.0.1"
# define PORT 23422
# define PORTSEND 23420
# define BUFSIZE 2048

using namespace std;
using Eigen::MatrixXd;
double N_I,N_O,N_HL;
float y_max,y_min;
float dataudp[10];
float senddata;
float Beta, Roll;
float windup = 5;
float gain_p = 10;
float gain_i = 0;
float gain_d = 5;
float prev_error = 0;
float int_error = 0;
float control = 0;
float Phio = 0;
float Phi1 = 0;
float Phi2 = 0;
float dt = .033;
float diff ,p_term ,i_term ,d_term ,current_error ,error ,P,Pdot ,Phi;

int main()

```

```

{
    // UDP
    SOCKET s;
    struct sockaddr_in server, si_other, send;
    int ss, slen, recv_len, slen1;
    char buf[BUFSIZE];
    char bufOut[BUFSIZE];
    WSADATA wsa;
    slen = sizeof(si_other);
    slen1 = sizeof(send);
    WSAStartup(MAKEWORD(2,2), &wsa);
    s = socket(AF_INET, SOCK_DGRAM, 0);
    ss = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(PORT);
    memset((char *) &send, 0, sizeof(send));
    send.sin_family = AF_INET;
    send.sin_port = htons(PORTSEND);
    send.sin_addr.S_un.S_addr = inet_addr(SERVER);
    bind(s, (struct sockaddr *)&server, sizeof(server));

    // Read Raw Data from txt file
    ifstream input("OfflineNN1.txt");
    input >> N_I >> N_O >> N_HL >> y_max >> y_min;
    int size = ((N_I+N_O)*N_HL);
    double data[size];
    int size1 = (N_I*N_HL);
    double Wl[size1];
    int size2 = (N_O*N_HL);

```



```

double W2[size2];
for (int i = 0; i < size; i++) {
    input >> data[i];
}

int N_II = N_I;
int N_OO = N_O;
int N_HLL = N_HL;
// Separate data for 1 Matrix
for (int i = 0; i < size1;i++){
    W1[i] = data[i];
}
// Separate data for 2 Matrix
for (int i = 0; i < size2;i++){
    W2[i] = data[i+(size1)];
}

// Get Weight Matrix
MatrixXd W_I_HL_NN1(N_II,N_HLL);
MatrixXd W_HL_O_NN1(N_HLL,N_OO);
W_I_HL_NN1 =
Eigen::Map<MatrixXd>(W1,N_II,N_HLL);
W_HL_O_NN1 =
Eigen::Map<MatrixXd>(W2,N_HLL,N_OO);
cout << "Matrix_1_\n" << W_I_HL_NN1 << endl;
cout << "Matrix_2_\n" << W_HL_O_NN1 << endl;
cout << "Ready_for_FlightGear_\n" << endl;
// NN1
MatrixXd x(N_II,1); // Input to NN1
MatrixXd a_HL1(N_HLL,1),
y_HL1(N_HLL,1),

```

```

a_out(N_OO,1) ,
y_out(N_OO,1);

float cP = 0;

for (int i = 0; i < 1000;i++){
    if ((i > 200 ) && (i < 400))
    {
        cP = 10;
    }
    if (i > 400){
        cP = 5;
    }
    fflush(stdout); //clear
    // get data from flight gear
    recv_len = recvfrom(s, buf, BUFSIZE, 0,
        (struct sockaddr *) &si_other, &slen);
    sscanf(buf,"%f_%f_%f_\n",
    &dataudp[0],&dataudp[1],&dataudp[2]);
    // printf("%f %f %f \n",
    dataudp[0],dataudp[1],dataudp[2]);
    // send data to flight gear
    Phi = dataudp[0];
    P = dataudp[1];
    Beta = dataudp[2];
    Phi2 = cP - Phi;
    current_error = Phi2;
    int_error = int_error + (current_error * dt);
    if (int_error < -(windup))
    {

```

```

        int_error = -(windup);
    }
    else if(int_error > windup)
    {
        int_error = windup;
    }
    diff = ((current_error - prev_error) / dt);
    p_term = (gain_p * current_error);
    i_term = (gain_i * int_error);
    d_term = (gain_d * diff);
    control = p_term + i_term + d_term;
    prev_error = current_error;
x << control ,Beta ,P;
a_HL1 = -1*(W_I_HL_NN1.transpose()*x);
y_HL1 = 1/(1+a_HL1.array().exp());
a_out = -1*(W_HL_O_NN1.transpose()*y_HL1);
y_out = 1/(1+a_out.array().exp());    // Output of NNI
senddata = (y_out(0)*(y_max-y_min))+y_min;
printf ("%f_ %f_ %f_ %f\n",Phi , senddata , control ,Pdot);
sprintf (bufOut , "%f_ \n" , senddata);
sendto(ss , bufOut , strlen(bufOut) , 0 ,
(struct sockaddr *) &send , slen1);
Sleep(33);
}
}
end

```

```

#include <fstream>
#include <Eigen>
#include <stdio.h>

```

```

#include <winsock2.h>
#include <string.h>
#include <stdlib.h>
#include <sstream>
#include <Windows.h>

# define SERVER "127.0.0.1"
# define PORT 23422
# define PORTSEND 23420
# define BUFSIZE 2048

using namespace std;
using Eigen::MatrixXd;
double N_I,N_O,N_HL;
float y_max1,y_min1,y_max2,y_min2;
float dataudp[10];
float senddata;
float windup = 5;
float gain_p = 10;
float gain_i = 0;
float gain_d = 5;
float prev_error = 0;
float int_error = 0 ;
float control = 0 ;
float dt = .033;
float outNN1, outNN2 = 0;
float diff ,p_term ,i_term ,d_term ,
current_error ,error ,P,Beta ,Pdot ,Phi;

int main()

```

```

{
    // UDP
    SOCKET s;
    struct sockaddr_in server, si_other, send;
    int ss, slen, recv_len, slen1;
    char buf[BUFSIZE];
    char bufOut[BUFSIZE];
    WSADATA wsa;
    slen = sizeof(si_other);
    slen1 = sizeof(send);
    WSAStartup(MAKEWORD(2,2), &wsa);
    s = socket(AF_INET, SOCK_DGRAM, 0);
    ss = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(PORT);
    memset((char *) &send, 0, sizeof(send));
    send.sin_family = AF_INET;
    send.sin_port = htons(PORTSEND);
    send.sin_addr.S_un.S_addr = inet_addr(SERVER);
    bind(s, (struct sockaddr *)&server, sizeof(server));

    // Read Raw Data from txt file
    ifstream input("OfflineNN1.txt");
    input >> N_I >> N_O >> N_HL >> y_max1 >> y_min1;
    int size = ((N_I+N_O)*N_HL);
    double data[size];
    int size1 = (N_I*N_HL);
    double W1[size1];
    int size2 = (N_O*N_HL);

```

```

double W2[size2];
for (int i = 0; i < size; i++) {
    input >> data[i];
}

int N_II = N_I;
int N_OO = N_O;
int N_HLL = N_HL;
// Separate data for 1 Matrix
for (int i = 0; i < size1;i++){
    W1[i] = data[i];
}
// Separate data for 2 Matrix
for (int i = 0; i < size2;i++){
    W2[i] = data[i+(size1)];
}

MatrixXd W_I_HL_NN2(N_II ,N_HLL);
MatrixXd W_HL_O_NN2(N_HLL,N_OO);

// Get Weight Matrix
MatrixXd W_I_HL_NN1(N_II ,N_HLL);
MatrixXd W_HL_O_NN1(N_HLL,N_OO);
W_I_HL_NN1 = Eigen::Map<MatrixXd>(W1, N_II ,N_HLL);
W_HL_O_NN1 = Eigen::Map<MatrixXd>(W2,N_HLL,N_OO);
W_I_HL_NN2 = Eigen::Map<MatrixXd>(W1, N_II ,N_HLL);
W_HL_O_NN2 = Eigen::Map<MatrixXd>(W2,N_HLL,N_OO);
    cout << "Matrix_1_\n" << W_I_HL_NN1 << endl;
    cout << "Matrix_2_\n" << W_HL_O_NN1 << endl;
// NN1
MatrixXd x(N_II ,1); // Input to NN1

```

```

MatrixXd a_HL1(N_HLL,1),y_HL1(N_HLL,1),
a_out(N_OO,1),y_out(N_OO,1);
MatrixXd a_HL2(N_HLL,1),y_HL2(N_HLL,1),
a_out2(N_OO,1),y_out2(N_OO,1);
ifstream input2("OfflineNN2.txt");
input2 >> N_I >> N_O >> N_HL
    >> y_max2 >> y_min2;
int sizenn2 = ((N_I+N_O)*N_HL);
double datann2[sizenn2];
int size1nn2 = (N_I*N_HL);
double W1nn2[size1nn2];
int size2nn2 = (N_O*N_HL);
double W2nn2[size2nn2];
for (int i = 0; i < sizenn2; i++) {
    input2 >> datann2[i];
}
// Separate data for 1 Matrix
for (int i = 0; i < size1nn2;i++){
    W1nn2[i] = datann2[i];
}
// Separate data for 2 Matrix
for (int i = 0; i < size2nn2;i++){
    W2nn2[i] = datann2[i+(size1nn2)];
}

W_I_HL_NN2 = Eigen::Map<MatrixXd>(W1nn2,N_HL,N_HLL);
W_HL_O_NN2 = Eigen::Map<MatrixXd>(W2nn2,N_HLL,N_OO);
cout << "Matrix_3_\n" << W_I_HL_NN2 << endl;
cout << "Matrix_4_\n" << W_HL_O_NN2 << endl;
cout << "Ready_for_FlightGear_\n" << endl;

```

```

float cP = 0;

for (int i = 0; i < 10000;i++){
    if ((i > 200 ) && (i < 400))
    {
        cP = 10;
    }
    if (i > 400){
        cP = 5;
    }
    fflush(stdout); //clear
    // get data from flight gear
    recv_len = recvfrom(s, buf, BUFSIZE, 0,
    (struct sockaddr *) &si_other , &slen);
    sscanf(buf ,"%f_ %f_ %f_ %f_ \n",&dataudp[0] ,
    &dataudp[1],&dataudp[2],&dataudp[3]);
    // printf("%f %f %f \n",
    dataudp[0],dataudp[1],dataudp[2]);
    // send data to flight gear
    Phi = dataudp[0];
    P = dataudp[1];
    Beta = dataudp[2];
    current_error = cP - Phi;
    int_error = int_error + (current_error * dt);
    diff = ((current_error - prev_error) / dt);
    p_term = (gain_p * current_error);
    i_term = (gain_i * int_error);
    d_term = (gain_d * diff);
    control = p_term + i_term + d_term;

```



```

prev_error = current_error;
x << control ,Beta ,P;
a_HL1 = -1*(W_I_HL_NN1.transpose()*x);
y_HL1 = 1/(1+a_HL1.array().exp());
a_out = -1*(W_HL_O_NN1.transpose()*y_HL1);
y_out = 1/(1+a_out.array().exp());    ///output NN1
a_HL2 = -1*(W_I_HL_NN2.transpose()*x);
y_HL2 = 1/(1+a_HL2.array().exp());
a_out2 = -1*(W_HL_O_NN2.transpose()*y_HL2);
y_out2 = 1/(1+a_out2.array().exp());    ///outputtNN2
outNN1 = (y_out(0)*(y_max1-y_min1))+y_min1;
outNN2 = (y_out2(0)*(y_max2-y_min2))+y_min2;
senddata = outNN1+outNN2;
printf ("%f_ %f_ %f_ %f\n",Phi , senddata , control ,Pdot);
sprintf (bufOut , "%f_ \n" , senddata);
sendto(ss , bufOut , strlen(bufOut) , 0 ,
      (struct sockaddr *) &send , slen1);
Sleep(33);
}
}

```

9. References

- [1] A. J. Kalise B. S. Kim. Non-linear flight control using neural networks. *AIAA Journal of Guidance, Control, and Dynamics*, 20(1):26–33, 1997.
- [2] Mark B. Tischler Bernard Mettler and Takeo Kanade. System identification modeling of a small-scale rotorcraft for flight control design. AIAA, October 2001.
- [3] Bhandari S. Colgren R. et al Brown, N. Modular wireless avionics system for autonomous uavs. AIAA, August 2006.
- [4] D. J. Bugajski. A dynamic inversion based control law with application to the high angle of attack research vehicle. AIAA Guidance, Navigation, and Control Conference and Exhibit. AIAA, 1990.
- [5] J. Valasek C. Restrepo. Structured adaptive model inversion controller for mars atmospheric flight. *Journal of Guidance, Control, and Dynamics*, 31(4):937–953, July 2008.
- [6] J. Valasek D. Ito, J. Georgie. Re-entry vehicle flight control design guidelines: Dynamic inversion. *NASA Johnson Space Center*, NASA TP-2002-210771, March 2002.
- [7] Y. Kim D. Shin. Reconfigurable flight control system design using adaptive neural networks. *IEEE Transactions on Control Systems Technology*, 12(1):87–100, November 2004.
- [8] Alison A. Proctor et al. Eric N. Johnson. Development and test of highly autonomous unmanned aerial vehicles. AIAA, December 2004.
- [9] G. Sachs F. Holzapfel. Dynamic inversion based control concept with application to an unmanned aerial vehicle. Proceedings of AIAA Guidance, Navigation, and Control Conference. AIAA, August 2004.
- [10] C. M. Ha. Echo state networks: Appeal and challenges. *Journal of Control, Guidance, and Dynamics*, 18(4), July 1995.
- [11] et al. J. F. Haffner, N. T. Meyrer. A multi-layer perceptron replaces a feedback linearization controller in a non-linear servomechanism. AIAA, May 1998.
- [12] K. A. Wise J. S. Brinker. Stability and flying qualities robustness of a dynamic inversion aircraft control law. *Journal of Guidance, Control, and Dynamics*, 19(6), 1996.
- [13] M. R. Akella J. Valasek. Adaptive dynamic inversion control of linear plants with control position constraints. *IEEE Transactions on Control Systems Technology*, 20(4), July 2012.

- [14] Eric N. Johnson and Daniel P. Schrage. System integration and operation of a research unmanned aerial vehicle. AIAA, January 2004.
- [15] K. Parthasarathy K. S. Narendra. Identification and control of dynamical systems using neural networks. volume 1, pages 4–27. AIAA, August 1990.
- [16] M. Kincheloe M. R. Napolitano. On-line learning neural-network controllers for autopilot systems. *Journal of Guidance, Control, and Dynamics*, 33(6), November 1995.
- [17] S. Bhandari M. Rose. Development and validation of flight dynamics model of a uav airplane. AIAA, June 2012.
- [18] R. Colgren M. Sadraey. Robust non-linear controller design for a complete uav mission. AIAA, August 2006.
- [19] R. San Martin. Neural networks training architecture for uav modelling. WAC, July 2006.
- [20] M.B. McFarland and A.J. Calise. Robust adaptive control of nonlinear systems using neural networks. IEEE, September 1997.
- [21] E. Morelli. System identification programs for aircraft (sidpac). AIAA, August 2002.
- [22] B. Mroton. Stability of dynamic inversion control laws applied to non-linear aircraft pitch-axis models. *University of Minnesota, Institute for Mathematics and its Applications*.
- [23] S. Bhandari N. Anderson. Flight-testing of a uav aircraft for autonomous operation using piccolo ii autopilot. AIAA, August 2008.
- [24] R. C. Nelson. *Flight Stability and Automatic Control*, chapter 3. McGraw Hill Higher Education, Boston, Massachusetts, 1998.
- [25] D. Prokhorov. Neural networks approach to aiaa aircraft control design challenge. July 2005.
- [26] V.R. Puttige. *Neural Network Based Adaptive Control For Autonomous FLight of Fixed Wing Unmanned Aerial Vehicles*. PhD thesis, University of New South Wales, 2008.
- [27] and Greg J. Schulein Rendy P. Cheng, M. B. Tischler. R-max helicopter state-space model identification for hover and forward flight. AIAA, April 2006.
- [28] R. Rojas. *Neural Networks: A Systematic Introduction*, page 336. Springer, 1996.
- [29] J. Roskam. Airplane flight dynamics and automatic flight controls. *DAR Corporation*, 1998.

- [30] A. Bettadapura S. Bhandari, O. Dadian. Avionics system for uav flight controls research. AIAA, August 2013.
- [31] A. Samuel S. Bhandari and R. Colgren. R-max helicopter state-space model identification for hover and forward flight. AIAA, May 2006.
- [32] D. Tang S. Bhandari, A. Raheja. Modeling and control of uavs using neural networks. volume 24. AIAA, August 2012.
- [33] O. Dadian S. Bhandari, A. Raheja. Nonlinear control of uavs using multi-layer perceptrons with off-line and on-line learning. Proceedings of the American Control Conference. IEEE, June 2014.
- [34] Y. Kim T. Lee. Non-linear adaptive flight control using backstepping and neural networks controller. volume 24, July 2001.
- [35] B. Mettler V. Gavrillets, I. Martinos and E. Feron. Control logic for automated aerobatic flight of a miniature helicopter. AIAA, August 2002.
- [36] H. Markram W. Maass, T. Natschlager. A fresh look at real-time computation in generic recurrent neural circuits. Institute for Theoretical Computer Science, 2002.
- [37] Walter H. Pitts Warren S. McCulloch. Non-linear flight control using neural networks. *Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.
- [38] Packard A. Wu, F. and G. Balas. Systematic gain scheduling control design: A missile autopilot example. AIAA, September 2002.