Neural Adaptive Controller For Non-Linear Flight

Ohanes Dadian[[1]](#footnote-1), Subodh Bhandari2, Amar Raheja3, Ajay Bettadapura4

California State Polytechnic University, Pomona, CA, 91066

# This paper showcases National Science Foundation (NSF) funded research conducted by the Computer Science and Aerospace Engineering departments at Cal Poly Pomona. The effort focused on the development of adaptive flight controllers for unmanned aerial vehicles. This included a multi-layer perceptron (MLP) and an echo state network (ESN). The MLP is used for the offline model, while the ESN is used for the online model. The MLP will be the focal point as its adaptive and recurrent behavior allows it to naturally adhere to classical control laws, complete with a feedback loop. The ESN uses a supervised temporal approach to machine learning. This makes it a choice candidate for solving problems in dynamical systems, such as flight controllers. The MLP will be used primarily for error correction. The ESN is sparsely connected and holds 12 neurons in the hidden layer, with a single input and output signal. The hidden layer acts as a reservoir, as it exhibits a 40% leak rate of current knowledge. The network’s neurons are activated using a hyperbolic tangent function, whose output feeds naturally into our stochastic gradient decent based training method. The method utilizes the least mean squares algorithm for producing the error term between the teacher and output signal. There is no need for weight adjustment as the output signal is driven back into the network and used to bais the neuron states. The initial prototype was developed in MATLAB, while the production system was developed in C++. Both versions have been proven Lyapunov stable. Testing occurred through Software-in-the-Loop, using a simulated model of our TwinEngine aircraft and the FlightGear simulator.

# Nomenclature

*ANN* = Artificial Neural Network

*AIAA* = American Institute of Aeronautics and Astronautics

*ESN* = Echo State Network

*FDM* = Flight Dynamics Model

GNC = Guidance, Navigation, and Control

*GUI* = Graphical User Interface

*HIL* = Hardware-in-the-Loop

*ISR* = Intelligene, Survellience, and Reconaissance

*MLP* = Multi-Layer Perceptron

*NAS* = National Air Space

*SIL* = Software-in-the-Loop

# Introduction

U

manned Aerical Vehicles (UAVs) provides a platform for performing various military and civilian missions. This includes military applications such as intelligence, surveillance, and reconnaissance (ISR), battlefield damage assessment, and force protection. While civilian applications include remote sensing, scientific research, search and rescue missions, border patrol, surveillance of disaster-affected areas, aerial photography, aerial mapping for geotechnical survey, vegetation growth analysis, crop dusting, precision agriculture, and assessment of topographical changes. The UAV industry is the fastest growing sector of aerospace industries and the use of UAVs has been growing significantly for civilian applications. It is estimated that UAV spending will double over the next decade from current worldwide expenditures of $5.2 billion annually. Once the technology matures to integrate the UAVs into the National Airspace System (NAS), the domestic UAV market is expected to increase rapidly and that will not only create tens of thousands of jobs, but also save money for federal, state, and local governments and taxpayers by dramatically reducing the need for expensive manned airplane and helicopter for a number of missions.

As the role of UAVs and their level of autonomy increase, it will become imperative that design flight controllers that perform well over the entire flight envelope. [1]

Traditionally, flight controllers are designed using linearized flight dynamics models. However, controllers based on these models only work under certain flight conditions around which the aircraft has been linearized. UAVs are expected to fly over a wide range of this envelope, characterized by frequent changes in dynamic pressure. This may be remedied by the use of controller based on non-linear dynamic models. This is due to their ability for flying over the entire flight regime.

One popular method for handling parameter variations and nonlinearity in flight control is gain scheduling. [2] [3] Unfortunatley, this technique comes with some disadvantages. Frequent and rapid changes in controller gains may lead to instability. [4] Another drawback is the high design and implementation costs that increase the number of operating points. Some of these disadvantages can be overcome through the use of nonlinear controllers based on feedback linearization (NDI). One such technique is nonlinear dynamic inversion. That has become a popular choice for UAV control. [1]

NDI requires accurate knowledge of flight dynamics and the controller may not work if the dynamics model is inaccurate or the system is in non-minimum phase. Robust nonlinear controllers have been designed, however they only work well under bounded uncertainties and are computationally intensive.

For this reason, UAVs need an onboard algorithm for the calculation of controller parameters that take into account the changing flight dynamics and nonlinearity, while at the same time can handle the presence of disturbance, parameter uncertainties, modeling error, and unmolded dynamics. Neural network based controllers provide a solution to this problem.

Neural Networks are intelligent agents that utilize layers of software-based neurons and process data based on a connectionist approach to computation. The underlying architecture is roughly based on the structure of biological neural networks. Like many intelligent agents, neural networks are adaptive. In that they change their internal structure to fit the environment they are in. Neural Networks are generally used to find patterns in data when the relationship between the input and output are quite complex. Our main interest in these systems is to find a method to autonomously fly a raptor helicopter and twin-engine aircraft. In my research, into this topic, I looked into many neural networks. However, I am comparing two of them; a Recurrent Neural Network and a Feed-Forward Multi-layer Perceptron. The purpose of this comparison is to find the choice network to train the raptor against.

# Aerial Vehicle Platforms

The aerial vehicle platform being used for this research is the 12' Telemaster airplane, which is 90 inches long with wing span of 144 inches. The airplane is shown in Figure 1. The empty weight of the airplane is about 24 pounds with payload capacity of about 12 pounds. The airplane is equipped with a Piccolo II autopilot. However, we are developing a custom avionics system with open source architecture. The existing autopilots are closed source and are not suitable for the implementation of neural network based controllers.



Figure 1. 12” Telemaster

# Artificial Neural Network

In order to provide autonomous flight, an adaptive controller will need to be developed. As stated previously, an echo state neural network was chosen as the desired model. [5] The network uses a reservior model in the hidden layer, which is non-linear and sparsely connected. [6] This model exhibits supervised temporal behavior for performing machine learning. This allows for solving problems in dynamical systems, such as

flight controllers. The architecture is depicted in Figure 3. The network is made of an N dimensional reservior with a K dimensonal input layer and a L dimensonal output signal.

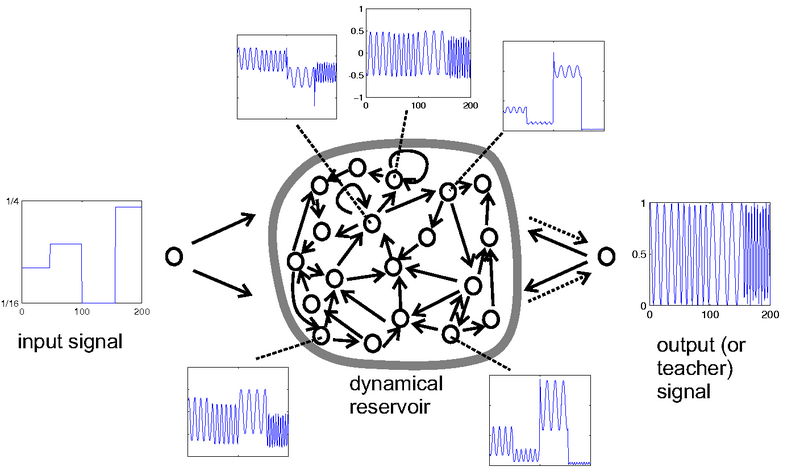
The neurons in the network are activated by an activation function. In this case, using a hyperbolic tangent formula (see Equation 1). This function is used when generating the reservior's current state. The previous state, as well as the input and teacher signals are factored into the input parameter of the tanhx function.

Figure 3. Echo State Network (via Scholarpedia.org)

Keep in mind that x(n) is an N dimensional state matrix. The output signal is generated using equation 3. This equation(g()) is also a sigmoid function, like the activation function. A logistic sigmoid, in this case (see equation 4). In order to properly generate output signal through Weiner-Hopf, a few state matrix transformations need to occur in computing the output weights.

After the neurons have been activated, a read-out phase occurs. This is where an output signal is generated and the output weights of the neurons are computed. The output signal is compared against the teacher signal. These signals are used during weight adjustment through linear regression. We are using a variant called the Wiener-Hopf method. This is a popular approach used for linear regression across applied mathematics. In the cases where integration is required of two-dimensional partial differential equations with mixed boundary conditions. A fourier transform is used to decompose a mathematical represenatation of the network signals frequencies depicting variations of the original formula. This essentially splits a function into two. As you will see, Weiner Hopf ultimately occurs by the multiplation of two matrices coorelated from the network's signals.

First, an extended state matrix (z) needs to be created. This is the concatenation of the reservior and input states (see Equation 5). This matrix is then filled row-wise into a state collection matrix. In a similar fashion, the desired outputs are put into row-wise matrix, D. The row-wise ordering is used in order to perform proper matrix multiplication during Weiner Hopf, with the correct dimensions. The transposition of this matrix is multiplied by itself to create a coorelation of the extended reservior states (see Equation 6). Next, a matrix representing the cross section of the cross-coorelation and desired outputs are computed (see Equation 7). With these results, the output weight matrix can be computed as seen in Equation 8.

This network shall contain 12 neurons in the reservior and a 40 percent leak rate. The leak rate pertains to the rate at which current knowledge is removed from the network.

The model prototype was developed in MATLAB. The model works as part of a SIMULINK model that passes flight parameters from my network. Its purpose is to perform gain scheduling on the flight parameters. The flight parameters include row, pitch and yaw. [7]

The MATLAB prototype was developed using R2012b and recently ported to R2015a. The code is driven by a high-level MATLAB function that is wrapped in a SIMULINK block. This allows for easy inclusion into the simulation model of our control system, as seen in Figure 6.

As mentioned earlier, the delta between the actual and target signals is computed in the output layer. This is accomplished by using the root mean square equation.

This error term is averaged over the N[y] dimensions of i(input) of the output signal, which is driven back into the system. The RMSE method allows for better human interpretability. Since it packs the data into small enough values without loosing the understanding of the difference between the signals eminating from the system.

In the future, we plan on testing a normalized root mean square, since it does not hold a dependency for the scaling of the target signal.

The neurons in the reservior (hidden layer) use leak-induced continuous values represented discretely over time. To update these neurons during online training, the following formula is used. Signal x(n) produces a real number and is a vector of the reservior neuron activations. tilda(x) is its update method occuring at time step n. The hyperbolic tangent is applied element-wis, while the vertical vector holds the input signal being driven into the reservior by the input layer. W denotes the weight matricies, for the input signal and the reservior. The alpha scalar denotes the leak rate being applied to the reservior.The following shows the leak rate currently being used by our model at the time of writing. After the neurons have been activated in the reservior, the output signal maybe be generated in the linear output layer. The output signal is defined as such. Wout is the output weight matrix. u(n) is the input signal, while x(n) is the reservior signal. The output weight matrix is multiplied against the vector of the input and reservior signals. As one might gauge, the weights are used to scale the values of the signals.

In order to account for modeling errors our control model needs to be trained online. The main benefit of this approach is to allow our system to continue learning while in flight. The main difference between offline and online is that the weight values update as the system operations. Like in offline training, the weights are updated using backpropagation. During training, it is important to ensure that convergence against the teacher signal is performed quickly without falling into a poor local minima. The better the local minima the less work our error correction model will have to do. Our intent is to use our simpler feed forward network for performing said error correction in our future dual model neural adaptive controller. With that being the case, keeping its work simple through good local minima will be in our benefit. It will keep our system design clean and allow for the recurrent model to provide the adaptability and robustness. The MATLAB based prototype runs off of text files for driving the input and output signals. These files hold matrices in comma separated value format (csv), as shown below.

The file contains a single signal with three values. The values being roll, pitch, and yaw. Once the input signal is loaded into the u vector, the reservoir is created. The reservoir is held in a 5x1 matrix, since the network holds 5 neurons and 1 output signal. The weights in the matrix are initially pseudo randomly created. The values scale between 0 and 1. These values determine the strength of the signals. After the input signal has been generated and the reservoir has been created, it is time to activate the neurons in the reservoir. The neurons are activated by a logistic sigmoid.

(1)

x(n) = (2)

(3)

u(n) is the input signal value tied to that particular neuron.

Once the neurons are activated in the reservoir, then network is ready to perform the read-out phase. This is a linear method for performing the output signal, which will then be compared with the teacher to perform weight adjustment. The following function is used to generate the signal.

(4)

This performs a matrix multiplication with the output weight matrix and the reservoir matrix. The reservoir matrix holds the values of the activated neurons. The output signal (Y) will be a 1x6 matrix. It will hold the activated values and input signal scaled by the output weight matrix.

This output signal is compared against a teacher signal. This signal is generated using flight data that is fed to it. In our case, we had our pilot, Hovig Yaralian, fly our 12 foot telemaster. The data from this flight was composed as a series of . The data was formatted into an input file as described previously. Using real-world flight data allows us to train our model without the need for a simulated flight dynamics model (FDM). This allows for a simple software-driven training environment.

Before we can feed the output weight matrix (Wout) back into the network with the input signal (u), stabilization needs to occur. This will allow us to keep our model stable, keeping a strong signal-to-noise ratio between the neural connections. A matrix with very large weights generally result in an unstable model. This is due to the fact that it would be very sensitive to small variations in the input signal.

This issue may be overcome by the use of regularization. Within the spectrum of machine learning, this consists penalizing large weights in order to prevent overfitting. The variant of regularization we employ is ridge regression. This is the combination of linear regression and L2 regularization. With its inclusion, the output weight matrix computation now looks like the following.

(5)

(6)

Where w is the cooresponding weight, b is the bias, and d is the number neurons in the reservoir. This now adds the concept of weight decay into our model. This a comprmise between having a small training error and small output weights. This compromise can be biased by the beta constant being multiplied against the summation. Of course, computing the delta between the z and y signals requires linear regression through matrix multiplication. First, we need an extended system state z(n). This state matrix is comprised of the input signal and neurons, .

Once acquiring the extended system state, the extended system states are filled row-wise into a state matrix S, ready for linear regression. The teacher signal is likewise filled row-wise into a state matrix D. A correlation matrix is then created, . Also, a cross-coorrelation matrix is created to depict the neuron states against the desired outputs. . Then we use Weiner-Hopf method to compute Wout.

(7)

This matrtix is now used to compute the output signal and driven back into the reservoir. For the purpouses of software-in-the-loop, we use a consistent input signal with an ever-changing output signal.This is the target signal mapped against the input signal for roll doublet, before training.

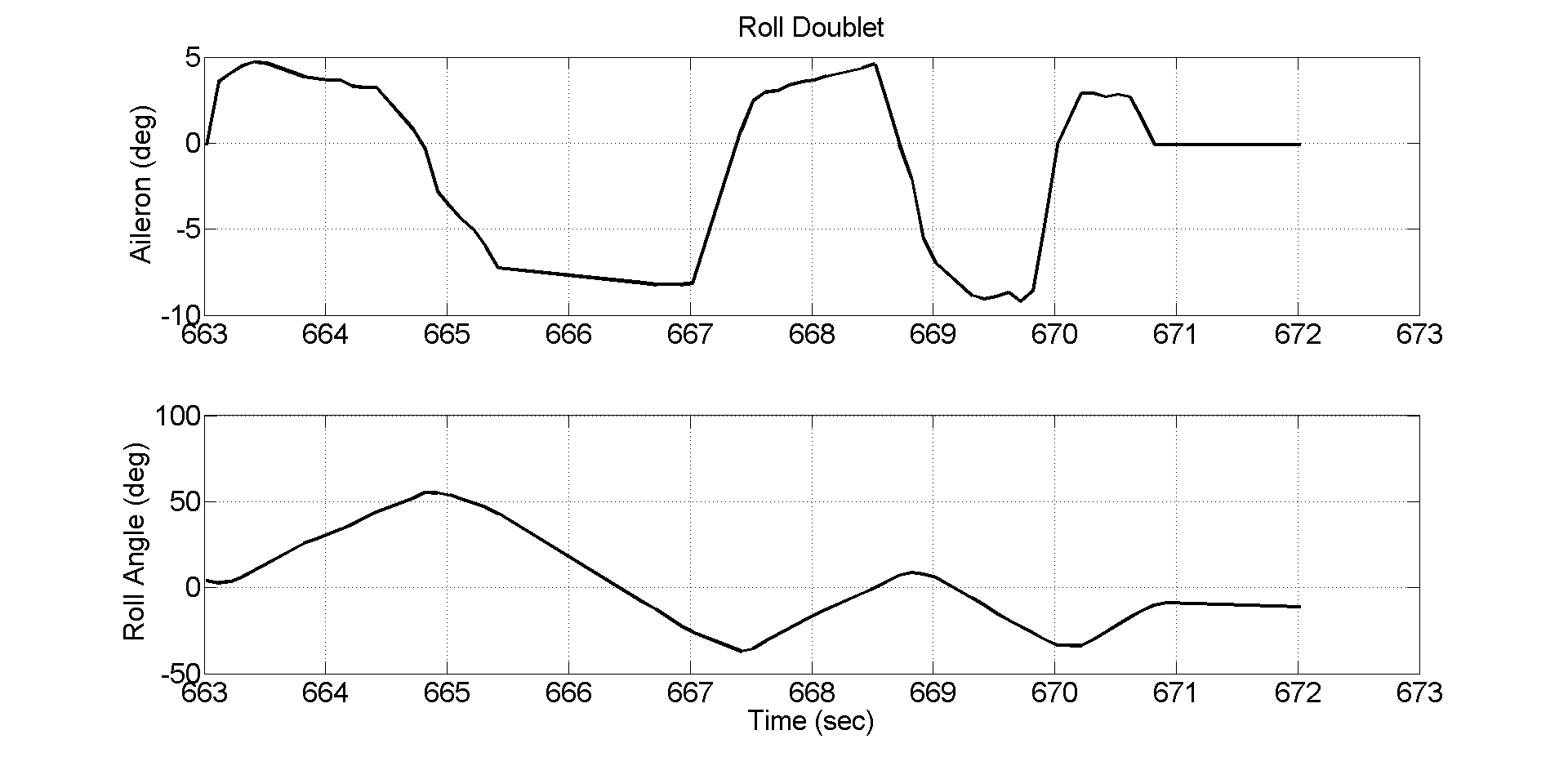


Figure 4. Roll Doublet Over Time

# Flight Dynamics Model

The dynamics model predicts the behavior of the aircraft following control inputs. It is developed from three force and three moment equations that describe the rigid body motion of the aircraft [18], [19].

 (8)

 (9)

 (10)

 (11)

 (12)

 (13)

where *c1*, c2*,* et cetera are functions of moments of inertias [10], *u*, *v*, and *w* translational velocities, *p*, *q*, and *r* are rotational velocities, *L*, *M*, and *N* and *Fx*, *Fy*, and *Fz* are the moments and forces acting on the airplane, *m* is the mass of the airplane, and *φ*, *θ*, and *ψ* are Euler angular rates [18].

To completely describe the aircraft motion, the three kinematic equations given below are required.

 (14)

 (15)

 (16)

The nonlinear equations given above are not suitable for control system design. These equations can be reformulated in the form given below [10].

 (18)

 (19)

 (20)

where , , , , et cetera are stability and control derivatives, *β* is the sideslip angle, *q* is the dynamic pressure, *b* is the wing span and *u0* is the forward velocity [18], [19].

All of the non-linear equations used for this research can be found in [10]. These equations are suitable for use with non-linear dynamic inversion.

Using these equations, a non-linear simulation model was developed for the airplane. Athena Vortex Lattice (AVL) software was used in the determination of stability and control derivatives required for the model [20]. The developed model was verified using flight data. Figure 2 shows the comparison of the flight data with the simulation results. The airplane was flown for doublet inputs in the pitch, roll, and yaw axes. It is seen that the simulation results compare well with the flight data for roll rate and yaw rate. Similar results were obtained for pitch responses.



Figure 5. Flight Dynamics over Time

The general non-linear equation can be written in the following form:

 (21)

For the aircraft problem, *x* is the state vector and *δ* is the control vector. A linearizing controller can be defined as

 (22)

 (23)

The above equation can be inverted to yield:

 (24)

As discussed below, a neural network can be used to realize this inversion, i.e., the network can be used to represent non-linear transformation for feedback linearization, eliminating the need for dynamic inversion [21].

# Identification and Control of 12' Telemaster airplane

This section talks about the neural network training and control of the 12' Telemaster airplane. The data required for off-line training was generated using the validated non-linear flight dynamics model discussed above. A sinusoidal input was applied to the model to generate the data. In order to learn, the network is trained on a large set of input and output data. The model generated data was divided to training data and validation data. The division occurs so that 70% of the data was used for training and 30% is used for validation. Figure 5 shows the implementation of the off-line trained network.

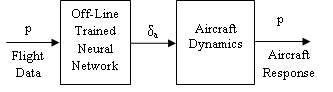


Figure 6. Implementation of Off-line Trained Network

The real-time flight data from the aircraft serves as input to the on-line training algorithm and controls outputs from the off-line trained network to provide targeted values for the on-line trainer as shown in Figure 6. In this system, the trained neural network and on-line trainer is the same object; so, the weights of the trained neural network are instantaneously updated on completion of the back-propagation process.

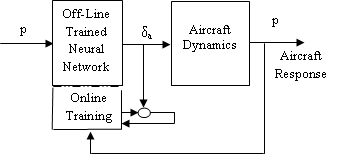


Figure 7. Implementation of On-Line Trained Network

In this work, the data required for on-line training is the simulation data obtained from the *FlightGear* Flight Simulator.

The neural networks were initially trained in MATLAB. However, for the SIL simulation and on-line training, MLP networks were ported from MATLAB to C. The C implementation ran a trained MLP on the QNX Neutrino Realtime Operating System (RTOS).

Figures 7 shows the responses of the off-line and on-line trained network compared with flight data for aileron. The flight data was collected for doublet inputs in aileron, rudder, and elevator for the 12' Telemaster airplane. It is seen that the response from the online trained network compares better with the flight data than the response from offline trained network only. It is also seen in Figure 7 that with the inclusion of on-line training, the response gets better as the training progresses.

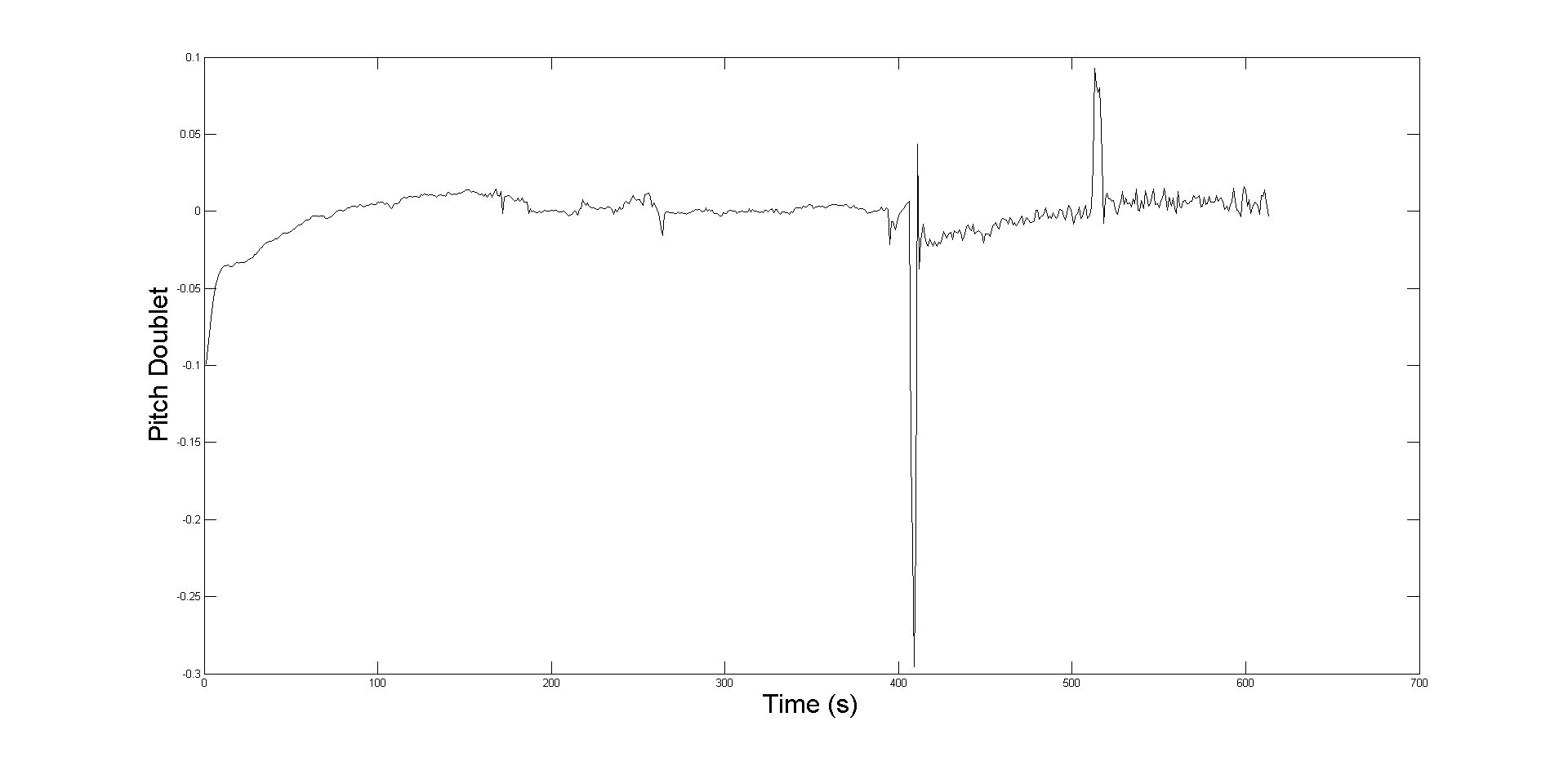


Figure 8. Flight Data vs. Neural Network (pitch doublet)

Figure 8 shows the airplane roll rate response collected during doublet maneuver and the responses from the neural network models. The comparison shows similar trends with online trained producing better results.

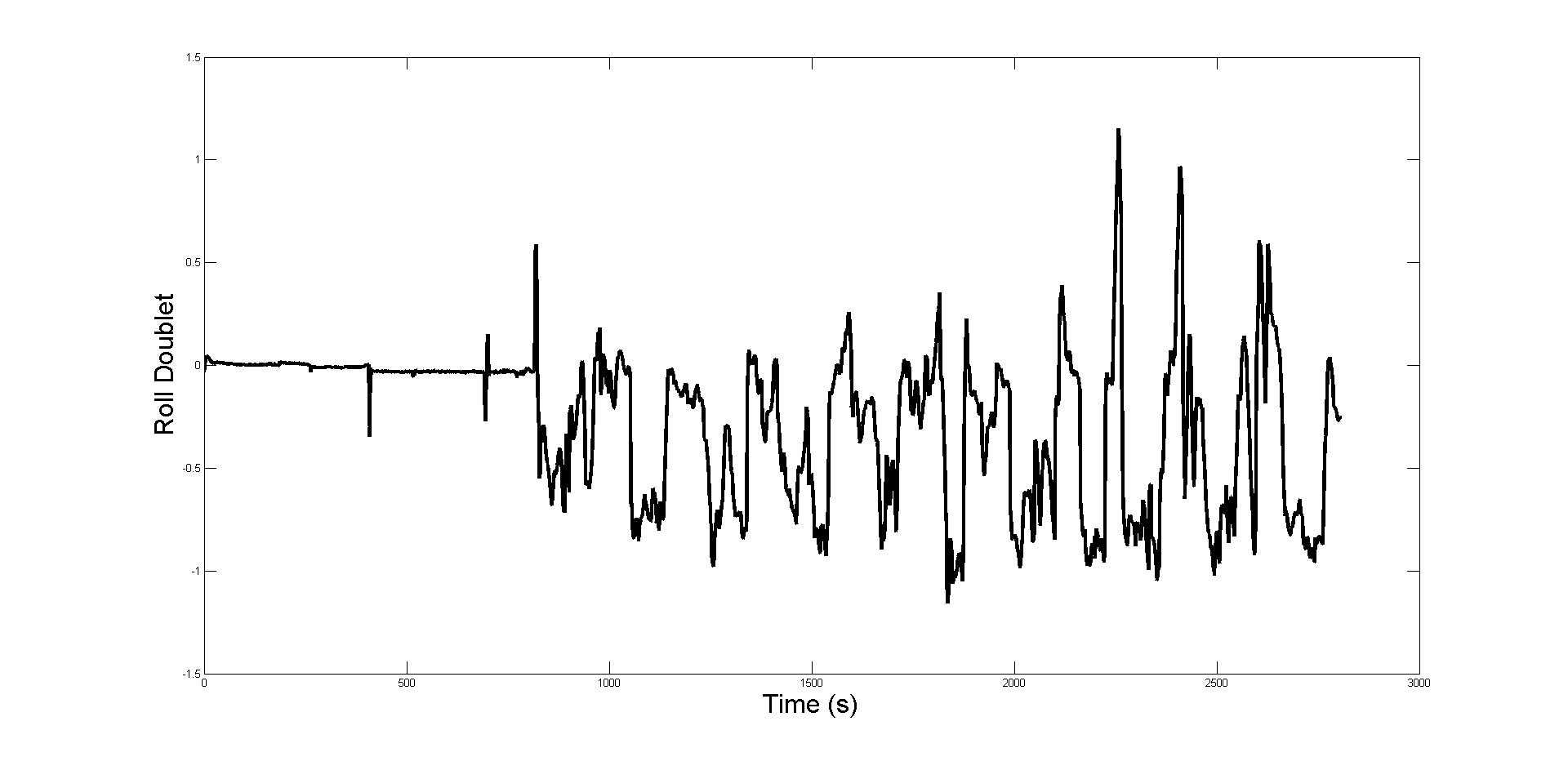


Figure 9. Flight Data vs. Neural Network Response (roll doublet)

Next, trained neural networks were implemented in a closed-loop system with feedback controllers included in the loop as shown in Figure 9.

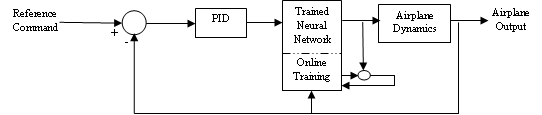


Figure 10. Neural Network Implemented in the Closed-Loop System

The feedback controller ensures stable closed-loop response and good transient characteristics. Figure 10 shows the closed-loop roll rate response of the airplane for a step command in roll rate.

Figure 11 shows the neural network implementation in a Software-in-the-Loop simulation environment, with the data for the online training obtained from *FlightGear* simulation of the aircraft.

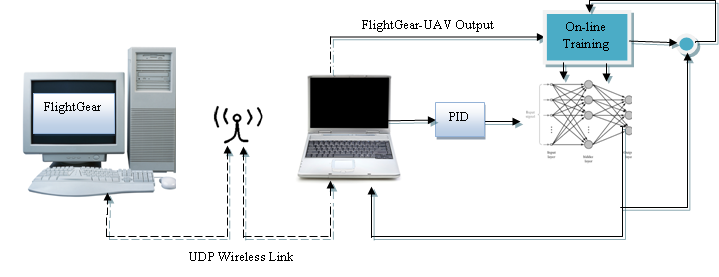


Figure 11. Software-in-the-Loop Simulation Block Diagram

The closed-loop system was then simulated with sensor noise included in the system. The sensor noise was modeled by a white noise of intensity of 0.5 deg./sec. This was done to test the robustness of the neural network controller when operating in non-ideal setting, much like in the real-world application of aircraft. Figure 12 shows the closed-loop response of the aircraft when subject to the white noise. It is seen that the neural network controller is able to track the command with reasonable overshoot and oscillation.

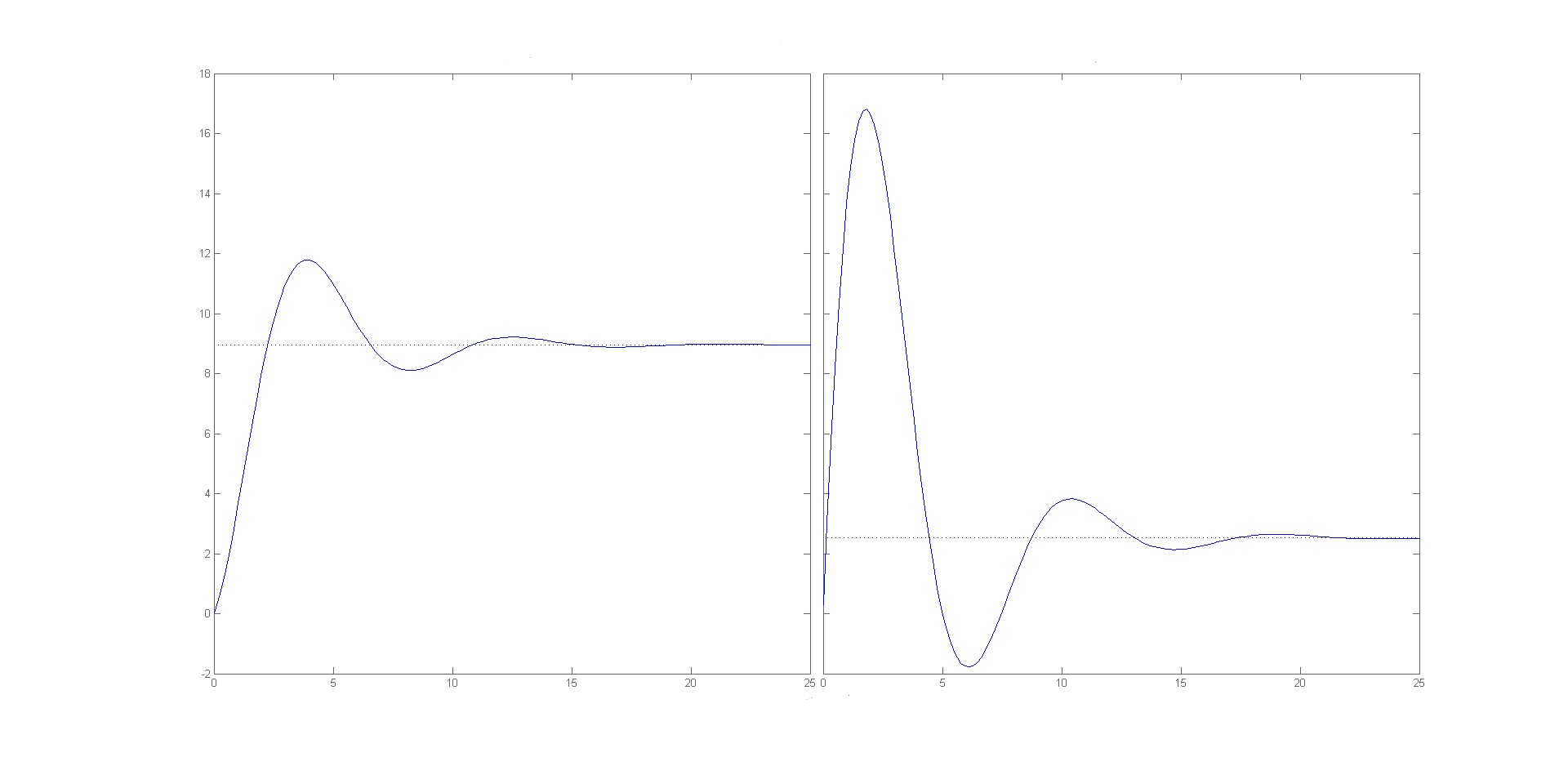


Figure 12. Response of the Aircraft in the Presence of Sensor Noise

# Results



Figure 13. ESN Plot

The ESN has been trained on flight data acquired from the Raptor 60 helicoptor and TwinEngine flight gear model. It was tested in software-in-the-loop simulation comprised of a series of PID controllers, servos, and state models. The MATLAB model was used in generating the plots in the this paper. From our tests, the output signal converges to the teacher signal on the first iteration of training (epoch). In the future, noise will be introduced to allow more of a descent in converging to better emulate environmental factors of real-time flight.

As can be seen in Figure 13, the input signal *u* veers towards from the target signal after the output signal (y) is driven back into the network.

# Conclusion

Cal Poly Pomona’s research effort to develop neural adaptive controllers has fostered strong results.The development of our MLP and ESN has helped push UAVflight controls research forward at our university. Most of the training occurred on the 12’ Telemaster aircraft. However, more UAVs are planned to be integrated in the future. The Telemaster pas proved to be a great asset, as its flight dyanmics model was the cornerstone of our software-in-the-loop simulation. The small error between the network signal and the actual signal seemed quite promising.

For future efforts, we plan on developing a plug-and-play flight controller atop of an FPGA. This will allows us to further modularize our open source avionics platform, presented at the 2013 Aerospace Sciences conference.

# Appendix

# Acknowledgments

The was supported by the National Science Foundation, under award no. 11023802. The authors would like to thank Hovig Yaralian, Luis Andrade, and Nigam Patel, undergraduate students in Aerosapce Engineering at Cal Poly, for their help their help with flight testing, and data integration of the flight dynamics model of the UAV. The authors would also like to thank Mr. Jim Cersai, the department technician, for his help with flight testing.

# References

|  |  |
| --- | --- |
| [1] | F. Holzapfel and G. Sachs, "Dynamic Inversion Based Control Concept with Application to an Unmanned Aerial Vehicle," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Providence, RI, 2004. |
| [2] | J. Shamma and J. R. Cloutier, "Gain-scheduled Mission Autopilot Design Using Linear Parameter-Varying Transformations," *Journal of Guidance, Control, and Dynamics,* vol. 16, no. 2, pp. 256-263, 1993. |
| [3] | F. Wu, A. Packard and G. Balas, "Systematic Gain Scheduling Control Design: A Missile Autopilot Example," *Asian Journal of Control,* vol. 4, no. 3, pp. 341-347, 2002. |
| [4] | S. Bhandari, O. Dadian and A. Bettadapura, "Avionics System for UAV Flight Controls Research," 2013. |
| [5] | S. Bhandari, O. Dadian and A. Bettadapura, "Nonlinear Control of UAVs using Multi-Layer Perceptrons with Off-line and On-line Learning," in *IEEE American Control Conference*, 2014. |
| [6] | S. Bhandari, O. Dadian and O. Bettadapura, "Search and Rescue using Unmanned Aerial Vehicles," 2015. |
| [7] | M. Lukosevicius, "A Practical to Applying Echo State Networks," 2012. |
| [8] | M. Samal, "Neural Network based Identification and Control of an Unmanned Helicopter," 2009. |
| [9] | T. Schulleys, "An Investigation into a Neural Network's performance based on the Number of Neurons in the Hidden Layer.," 2006. |
| [10] | R. San Martin, "Neural Networks Training Architecture for UAV Modeling," 2006. |
| [11] | V. Puttige, "Neural Networks Based Adaptive Control For Autonomous Flight of Fixed Wing Unmanned Aerical Vehicles," 2008. |
| [12] | Y. Hu, "The Application of an Annealing Recurrent Neural Network for Extremum Seeking Algorithm to Optimize UAV Tight Formation Flight," 2006. |
| [13] | K. S. Tsakalis and P. A. Ioannu, "Linear Time Varying Systems: Control and Adaptation," *Prentice Hall,* 1993. |

1. Graduate Student, Department of Computer Science, and AIAA Student Member.

   2 Professor, Department of Aerospace Engineering, and AIAA Professional Member.

   3 Professor, Department of Computer Science, and AIAA Professional Member.

   4 Undergraduate Student, and AIAA Student Member. [↑](#footnote-ref-1)