Colorado School of Mines Mechanical Engineering

Advanced Robot Controls - MEGN545

Project Part 1

Authors:
Aarushi Doctor - 10913401
Nick Taylor - 10920730

Class Personnel:
Dr. X. Zhang

March, 2024

Contents

1 Introduction & Problem Formulation

In this project, we aim to address these challenges by developing a robust control system for a quadrotor connected to a payload using neural network modeling techniques in Simulink. By leveraging the flexibility and adaptability of neural networks, we seek to create a control architecture capable of accurately modeling the complex dynamics of a quadrotor-payload system. We will develop a comprehensive model of the quadrotor and its connected payload, modeling the interactions between them. Utilizing neural network techniques, we will design a control system capable of effectively stabilizing and maneuvering the quadrotor-payload system. We will implement the developed control system in Simulink and conduct simulations to evaluate its performance under various modeling conditions.

2 Design Process

The purpose of this section is to overview the process undertaken to build the NN model.

2.1 Default Model

The NN model was originally designed by following the video tutorial provided in the project write up (https://www.youtube.com/watch?v=C5S_IHbP3xA).

Given the provided "classproject.slx" Simulink model, we started this assignment by running this model. This model outputs angle and position data into the MATLAB workspace. From there, this time-based angle and position data can be read into a neural network GUI, namely the "nnstart" MATLAB neural network GUI. This GUI allows for the neural network to be trained using the data obtained from running the provided "classproject.slx" Simulink model.

After training the NN model, this training data can be exported back into the MATLAB workspace as a structure. At this point, this trained neural network can be converted into a Simulink block using the commands netnodelay() and gensim(), which take inputs from the NN training data which was obtained from the nnstart MATLAB GUI. Because the nnstart MATLAB GUI can only train position OR angle data at one given time, this process has to be duplicated a second time so that we can obtain a NN Simulink block for angle AND position data.

Once these respective NN Simulink blocks are obtained, they can then be plugged into the "classproject.slx" Simulink model in the following orientation.

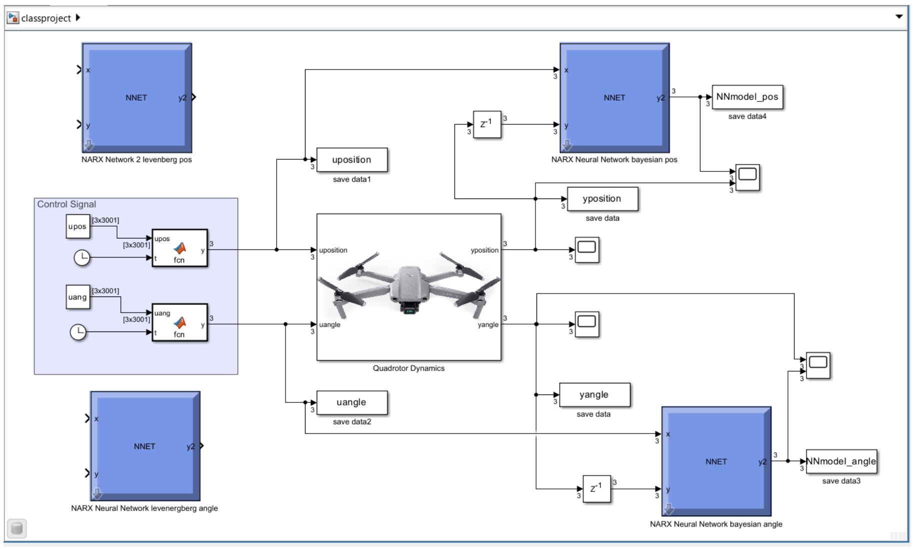

Figure 1: "classproject.slx" Simulink Model

2.2 Model Optimization

After running the Simulink model to achieve default functionality, we then explored the nnstart MATLAB GUI parameters in an attempt to tune for optimal NN parameters. We explored the 3 different training algorithms, as well as 3 different hidden layer sizes.

3 Results & Discussion

The purpose of this section is to present the results and acknowledge important implications drawn from the results of the simulation.

3.1 Default Model

After following the instructions as outlined in the Design Process section above, the training results for the NN model are shown below for both the angular and position NN models.

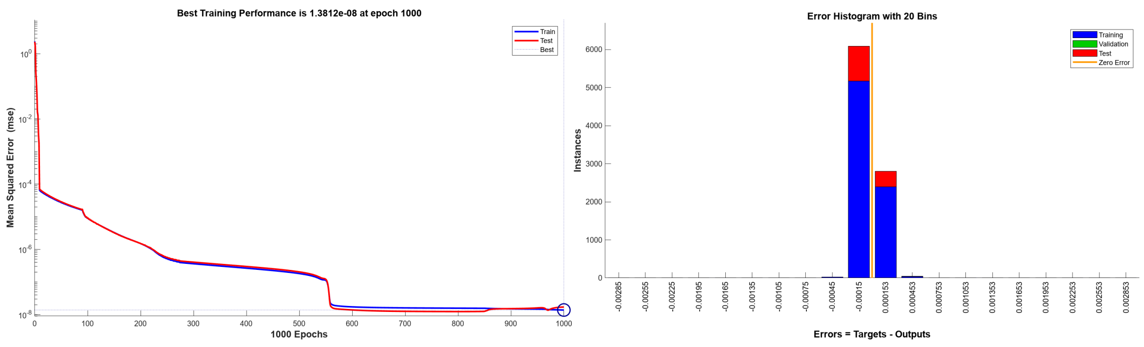

Figure 2: Angular NN Training Results

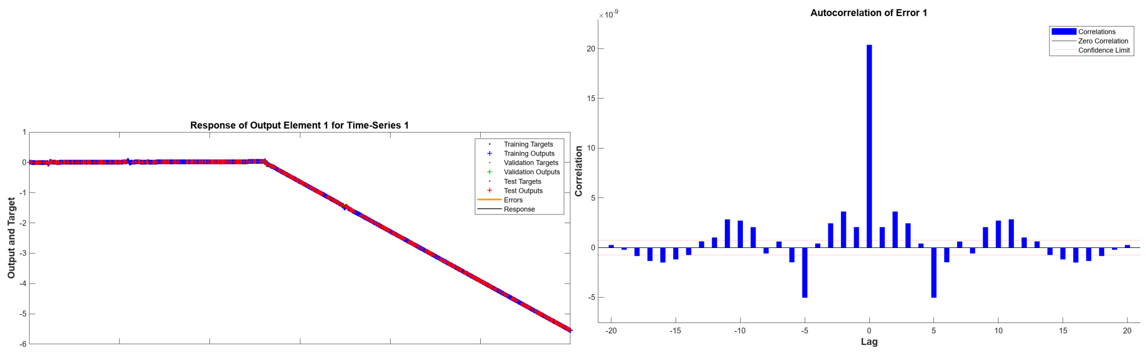

Figure 3: Angular NN Training Results

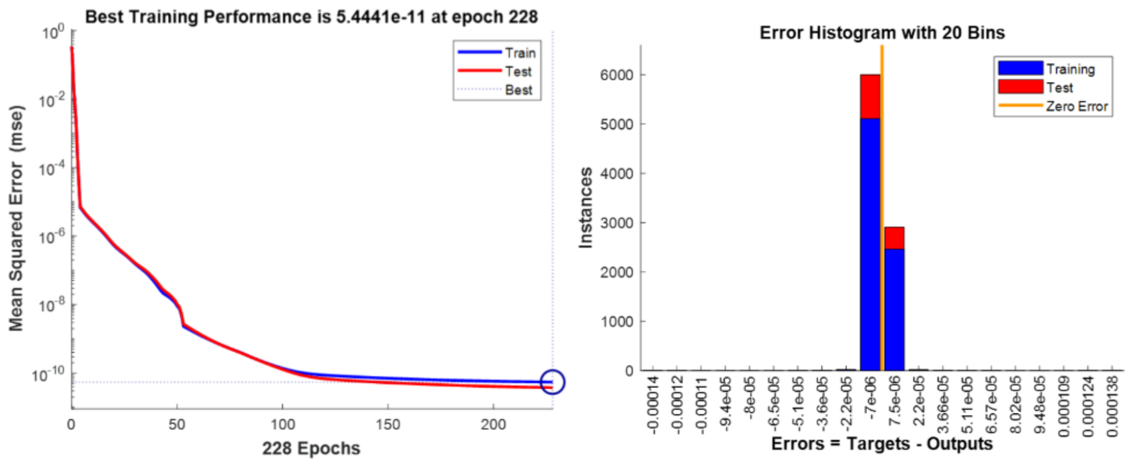

Figure 4: Position NN Training Results

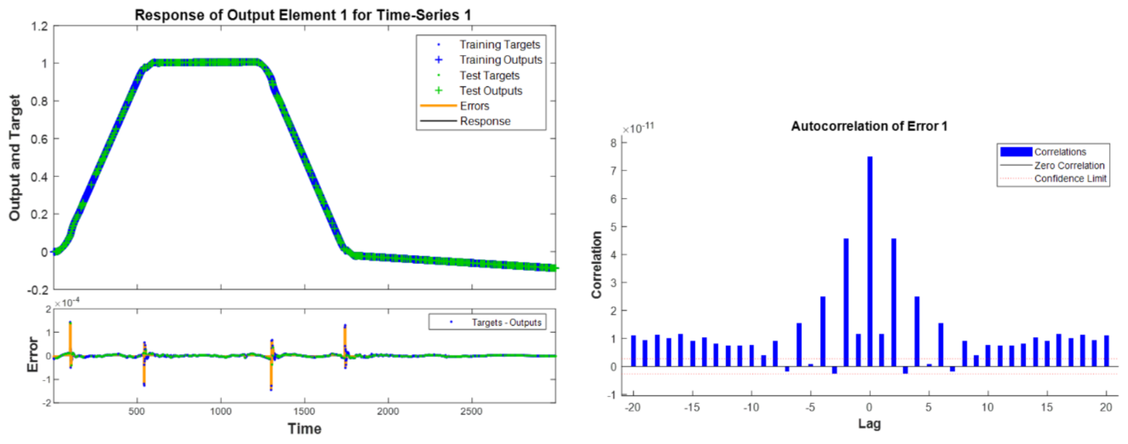

Figure 5: Position NN Training Results

Figure 6 and 7 below both show the desired control signal compared to the NN model actual output signal. Figure 6 shows angular data comparisons, while Figure 7 shows position data comparisons. These plots show successful validation of both NN models.

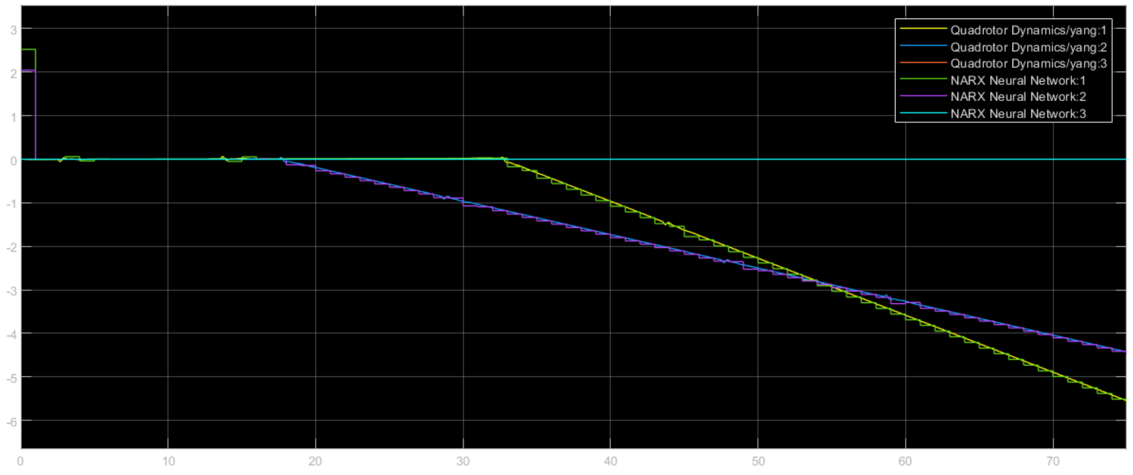

Figure 6: NN Model Angular Data Comparison

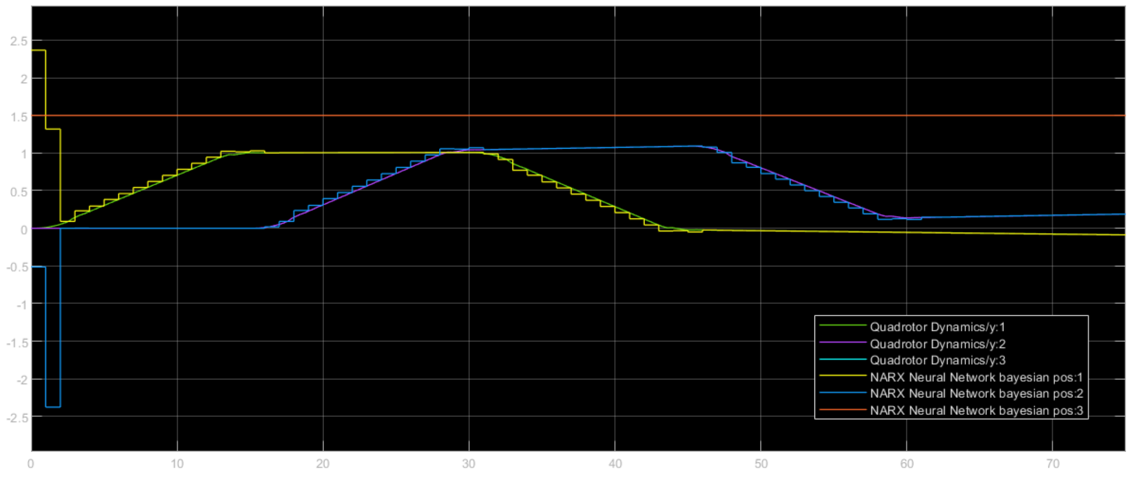

Figure 7: NN Model Position Data Comparison

3.2 Model Optimization

This section displays the results used to gauge the performance of each NN model using RMS error. We ran 5 different simulations, changing both the angular and position NN models within each simulation. The tables show how we varied the training method and hidden layer size within each simulation. The 3 tables show angular RMSE results, position RMSE results and holistic RMSE results. We determined what we call "holistic" RMSE results by simply taking the mean of the position and angular RMSE results for each respective simulation.

Training Method	Hidden Layer Size	Mean Position RMSE	Mean Angle RMSE
Bayesian	10	0.1780	0.1980
Levenberg-Marquardt	10	0.1415	0.1091
Scaled Conjugate Gradient	10	0.1026	0.1522
Bayesian	4	0.1957	0.2227
Bayesian	14	0.4352	0.1780

Table 1: Model Optimization Holistic Results

Training Method	Hidden Layer	X Pos. RMSE	Y Pos. RMSE	Z Pos. RMSE
Bayesian	10	0.3121	0.2819	4.6125e-14
Levenberg-Marquardt	10	0.1571	0.1703	4.8269e-13
Scaled Conjugate Gradient	10	0.1456	0.1623	1.2632e-13
Bayesian	4	0.2369	0.1544	4.6515e-14
Bayesian	14	0.4352	0.4351	4.6521e-14

Table 2: Model Optimization Position Results

Training Method	Hidden Layer	X Angle RMSE	Y Angle RMSE	Z Angle RMSE
Bayesian	10	0.2953	0.2387	0.00
Levenberg-Marquardt	10	0.2146	0.2098	0.00
Scaled Conjugate Gradient	10	0.2126	0.0918	0.00
Bayesian	4	0.2522	0.1931	0.00
Bayesian	14	0.2956	0.2385	0.00

Table 3: Model Optimization Angular Results

Surprisingly, the results show that Bayesian Regularization training method provide the largest RMSE values (worst NN model performance). Changing the hidden layer size to either 4 or 14 also proved to increase error and lower performance. These results show the smallest error results for the Scaled Conjugate Gradient and Levenberg-Marquardt training methods, both with a hidden layer size of 10.