Intelligent Control of Pneumatic Actuator Controlled Mechanical Arm

By
Patrick Musau

Department of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN 37235
Email: patrick.musau@vanderbilt.edu

Contents

Introduction

In recent years, Artificial Neural Networks have demonstrated an impressive ability to be used as frameworks for solving complex problems in numerous application domains [20]. In fact, the success of these models in contexts such as adaptive control, non-linear system identification [14], image and pattern recognition, function approximation, and machine translation, has led researchers to believe that neural networks posses the power to revolutionize the development of robust and intelligent control systems. Within this realm, the allure of neural network models is a result of their highly parallel structure which allows for distributed processing, a proficient ability to model complex non-linear systems [9], a capacity to handle large multivariate systems, and the capability of being encoded efficiently within hardware [7].

Generally within control system applications neural networks are typically utilized for function approximation applications where the task is to model some complex phenomena that lacks a formal description [5]. In fact, it has been demonstrated that neural networks are capable of approximating any continuous function to any desired level of accuracy [6]. Thus, neural networks are often used to model an extensive range of non-linear dynamics and promote the successful design and analysis of controllers for systems in which it is difficult to acquire a precise mathematical model of the plan considered for control [3]. This is particularly important since the majority of real-world systems are inherently non-linear in nature. Thus, within this realm the goal is to model and control the plant using data-driven control system methodology. It is within this context that neural networks display their prowess. By training a neural network on a set of recorded input-output traces of the system, we can obtain a model whose predictive behavior adheres to the behavior of the original system and using this model we can formulate control regimes that work very well in practice [17].

Thus, bearing the above in mind, in this work we present an investigation of the nonlinear functional mapping competence of neural networks in order to control a pneumatically driven mechanical arm [16].

1.1 Neural Network Preliminaries

However before we commence, what is a neural network? A neural network is a supervised machine learning model that is loosely modeled after the human brain. A neural network consists of a series of interconnected neurons where each neuron can be viewed as a processing element that reacts to the weighted sum of the inputs that it receives. The neurons are typically structured into three types of layers: an input layer, an output layer, and one or multiple hidden layers. Each connection between neurons is typically labeled with a real-valued weight that is determined during a training process that seeks to maximize the networks prediction accuracy [4]. The overall structure of the neural network is highly dependent on the particular architecture considered, and the are numerous architectures such as recurrent neural networks [13], radial basis function networks [18], long-term short-memory networks[15], and self organizing maps[11]. However in this manuscript we will focus on feed-forward neural networks since they are the most commonly utilized neural network within control systems applications. An example a such a network can be seen in Figure 1.1. In feed-forward neural networks (FNN) the connections

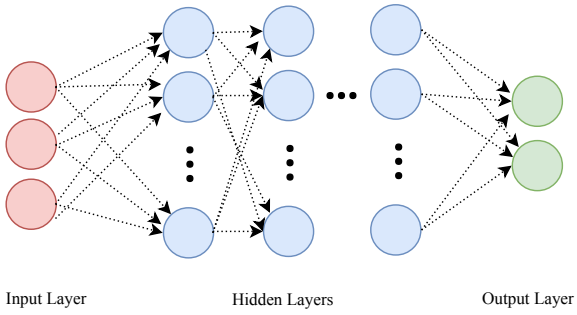

Figure 1.1: Simple Feed-forward Architecture

between neurons extend forward in a single direction. Each layer is fully connected to the next one, but connections between neurons within the same layer are prohibited. The output of a neuron is computed by calculating the weighted sum of the outputs of neurons in the previous layer and applying a non-linear activation function to that sum shifted by a bias. The action of a single neuron can be described by:

$$u_i = f(\sum_{j=1}^{n} w_{ij}x_j + b_i) \tag{1.1}$$

By connecting a series of neurons in this fashion, the operation of a neural network can be viewed as a function, $v : I^n \to O^m$, that maps an n-dimensional input, I^n, to an m-dimensional output, O^m [12]. For a neural network with L layers, where $1 \leq \ell \leq L$, let s_ℓ denote the size of the layer ℓ, s_1 denote the size of the input layer, $v_{\ell,j}$ denote the value of the j-th neuron of layer ℓ, b_ℓ be a bias vector of size $s_\ell \times 1$, and $U_\ell = [u_{\ell,1}, \ldots, u_{\ell,s_\ell}]^T$ be a column vector of neuron activations for layer ℓ ([19, 10]). For each layer $2 < \ell < L$, there is

an associated weight matrix, W_ℓ, of size $s_\ell \times s_{\ell-1}$. Thus, the neuron activation vector, U_ℓ, can be described by:

$$U_\ell = f(W_\ell \cdot U_{\ell-1} + b_l) \tag{1.2}$$

where the activation function $f(\cdot)$ is applied component-wise [10]. Given an input, U_1, the output of the whole network is given by applying rule (1.2) repeatedly until U_ℓ is calculated.

In order to train the neural network to approximate a given function of interest, we must tune the weights W_l, and biases b_l in equation (1.2) for each neuron in the network. This is typically done using algorithms [4] such as dynamic back-propagation and some form of gradient descent. In these schemes, the network is trained by minimizing the prediction error between the network's prediction computed using equation (1.2) and the corresponding recorded output, collected from the dynamical system. There are numerous optimization algorithms utilized in training FNNs and they have been largely successful in training networks to perform various complex tasks. A brief discussion of these techniques will be presented in the proceeding sections of this manuscript and we further refer readers to the following paper for an in depth discussion of these methods [8].

Objectives

Bearing the above in mind, the overall objective of this work is to design and use a feed-forward neural network in order to control the angular position of a mechanical arm driven by two pneumatic actuators called rubberators. To do this we must gather sufficient sensor data from an angular orientation encoder in order to train the neural network to map the angular positions to their corresponding voltage inputs so that the correct pressure is delivered to the pneumatic actuators. The experiments presented in this work demonstrate that a user can successfully utilize the network to control the mechanical arm. Figure 2.1 displays a block diagram representation of the system that our methods implemented.

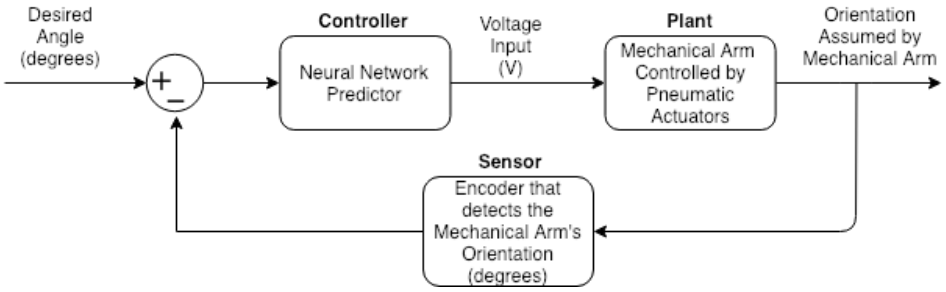

Figure 2.1: System schematic for the intelligent control of the mechanical arm.

The controller for our system will be a feed-forward neural network, the plant we seek to control is the mechanical arm, and the sensor is the angular position encoder. Each of the blocks in Figure 2.1 will be discussed in detail in the subsequent sections of this manuscript.

System Description

The mechanical arm considered in this work consists of two pneumatic actuators, called *rubberators*, that are connected to a hinge joint. The rubberators belong to a class of non-linear actuators called Mckibben actuators, and were designed to be artificial models of muscles for use in robotics applications [2]. These muscles consist of an internal bladder surrounded by a braided shell that is attached to fittings on either end [2]. When the internal bladder is pressurized, it expands and pushes against its inner surface thereby increasing its volume. The rubberator's braided shell possesses a high longitudinal stiffness, and due to this non-extensibility, the actuator shortens in proportion to the increase in volume increase [2]. This shortening produces tension that can be coupled to a mechanical arm in order to induce rotation [16]. A diagram of a pneumatic actuated mechanical arm can be seen in Figure 3.1. In this configuration, the angular position of the joint, given in

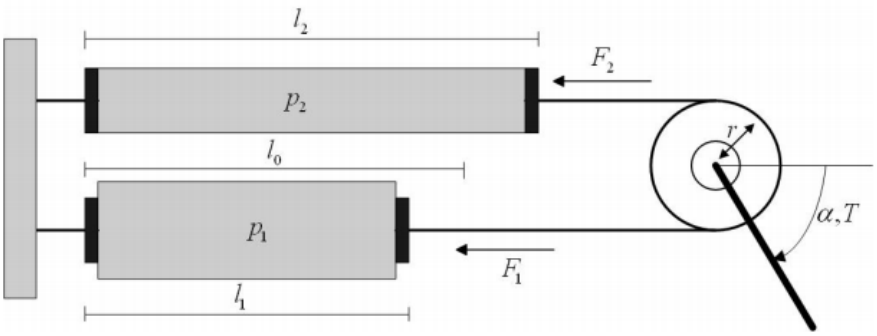

Figure 3.1: Diagram of Mckibben Actuated Mechanical Arm as shown in [16]

degrees, can be altered by changing the pressure in each arm. The pressure is given by a regulator and is encoded in volts. Initially each rubberator is inflated to $1.2V$, as given by the regulator, and the maximum pressure is equal to the input pressure when the regulator records a value of $2.5V$. In order to control the mechanical arm, one must alter the ratio of

the maximum pressure given by

$$\frac{\text{Current Voltage}}{\text{Maximum Voltage}} * \text{Pressure input} \tag{3.1}$$

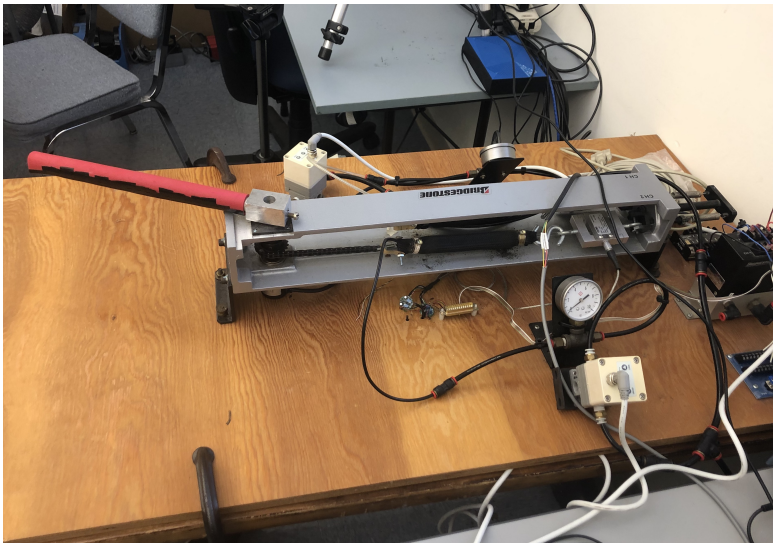

Figure 3.2: Mechanical Arm Experimental Setup

Thus the command to the Rubberators will be a change in voltage ΔV, where the pressure of one Rubberator is increased by ΔV and the other is decreased by ΔV. The change in pressure in each arm will cause the arm to rotate toward the direction of the Rubberator with a higher pressure. Depending on the sign of ΔP, the arm will rotate in either the "positive" or "negative" direction. Lastly the system has one feedback sensor, which consists of an encoder that captures the angular position of the mechanical arm measured in degrees. It is this sensor that will be used to collect data in order to create a neural network that maps desired angles to their corresponding voltage inputs.

Methods

4.1 Data Collection

To successfully train a neural network we needed to collect a sufficient amount of data. The data that we collected consisted of a set of system simulations using a variety of voltage input signals (ramp, step, sinusoidal, etc.) so that the network could learn to map joint angles to input voltages. Using the Simulink model shown in Figure A.6, we collected

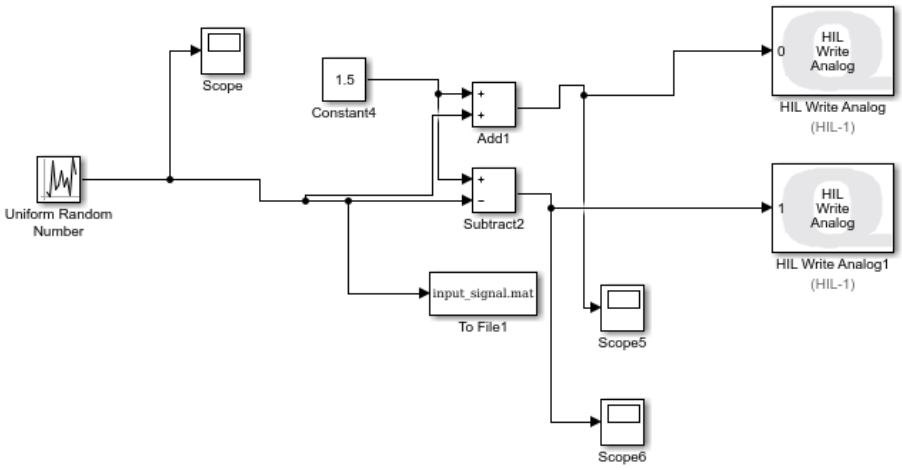

Figure 4.1: Simulink model of the data collection phase of the project

training data by providing the pneumatic actuators with a series of input voltages and recorded the value that the encoder registered for the mechanical arm's orientation. The voltage signals that we considered were generated using a sine wave with an amplitude of 1.4 volts, and a repeating sequence generator that produced signals between -1.4V and 1.4V

in increments of 0.2V. A plot of the system simulations in response to the input voltages can be seen in Figure A.1 and Figure A.2.

4.2 Neural Network Training

With the data that we collected in section 4.1, the next step in obtaining a successful neural network model was to tune the network's parameters (the weights and biases) so that it performed well on the desired angle to voltage input prediction task [1]. However, to asses how well the network was performing on the task, we must first define a function known as the *loss function*. The loss function provides a sense of the network's performance and it is often defined in terms of the mean squared error. Intuitively what is done, is that for every input in the training set we compute the network's prediction and compare it to the output we expect it to produce. Thus the loss function can be defined as:

$$L(y, \hat{y}) = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 \tag{4.1}$$

where m is the number of training examples, y_i is the output predicted by the network and $\hat{y}_i$ is the correct output from the training data [1]. The loss for an incorrect prediction carries a positive value while the loss for a correctly classified example is zero. Thus, the goal of training process is to minimize the loss function [1].

During the training process, we begin with a network with randomly generated weights and by changing these weights we are able to alter the value of the loss function. Thus the task of optimizing the loss function can be described as the problem of selecting the weights and biases such that we maximize the number of correct predictions given by the network [8]. There a numerous optimization algorithms within the research literature but the most common algorithms are a form of gradient descent. In gradient descent based regimes, one alters the network's weights and bias parameters in the direction of the negative gradient. This is done so that we traverse in the direction of the minimum of the loss function. In order to change the weights, they are updated using an algorithm known as dynamic back-propagation and by altering the weights in this fashion one can obtain a model that performs quite well. While this is a high level description of the training process, we refer readers to the following paper for an in depth discussion of the training process [4]. However in this manuscript, we allowed Matlab to handle the optimization of the network's parameters.

In our experiments, the training algorithm that we utilized was Batch Gradient Descent with Bayesian regularization. This was done using Matlab's nntool framework. Bayesian regularization is a method for ensuring that the network will generalize well on unseen inputs once the training process has completed. Since we wish to ensure that a user can map an arbitrary desired angle to a correct input voltage, we made use of this algorithm in order to obtain suitable performance. After 500 iterations the mean squared error of the network on the training data was 0.2501. Thus our network performs quite well on our

prediction task. However, we must note that in order to achieve this favorable performance a significant amount of data pre-processing was required. Unfortunately, as is often the case with data collection devices, the encoder that measured the mechanical arm's orientation produced readings that were off by 24.5 degrees. Thus, during the preliminary round of testing, the networks that we trained learned a model that was not physically realizable. However by shifting the data by 24.5 degrees, we were able to learn a successful model that produced voltage recommendations that were reliable. The networks performance on the training data as well as an independently generated set of testing data can be seen in Figure A.3 and Figure A.4.

4.3 Model Evaluation

Once the network achieved a suitable performance, we created a Simulink block that captured the networks behavior. Using this network we were able to simulate a user utilizing the system successfully, and we have provided a link to a visual demonstration of the system in the supplementary material of this paper[1]. Additonally, we created four different neural networks of various sizes to accomplish the task. The networks that I was responsible for training each had three layers. The first network had a total of 15 neurons in its hidden layers and the second neural network had 4 neurons. Both of the networks had a low mean squared error and I included them in the Simulink model as shown in Figure A.5. Finally, we simulated the system on sinusoidal inputs, impulse inputs, ramp inputs, and step inputs and we have included each test within the Simulink model. Thus we achieved our goal of allowing a user to map angles to input voltages for the mechanical arm system.

[1] A visual demonstration of our experiments, along with the data used to train the neural network can be found at: `https://github.com/pmusau17/NeuralNetworkControl`

Conclusion

The success of Artificial Neural Networks as frameworks for solving complex problems has led researchers to believe that these models will revolutionize the development of robust and intelligent systems in a diverse set of application domains. In this work we presented an investigation of the nonlinear functional mapping competence of neural networks in order to control a pneumatically driven mechanical arm. The results of our experiments demonstrate that we were successful in creating a system that allowed a user to specify a desired input angle to the system and induce this orientation on the mechanical arm. The networks that we trained had low mean squared errors and performed well on both the training data and an independent set of testing data aimed at assessing generalizations. Although the preliminary results of this paper are promising there are several interesting directions for future work including:

- Determining if the neural network models are robust to slight perturbations in their inputs,
- Determining if a network that makes use of both an angular position encoder and a digital camera is more accurate,
- A steady state and transient analysis of the system
- And lastly, considering other control regimes such as Neuro-Fuzzy Control.

References

[1] Vitaly Bushaev. *How do we 'train' neural networks ? – Towards Data Science*. 2017. URL: https://towardsdatascience.com/how-do-we-train-neural-networks-edd985562b73.

[2] Ching-Ping Chou and B. Hannaford. "Static and dynamic characteristics of McKibben pneumatic artificial muscles". In: *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. 1994, 281–286 vol.1. DOI: 10.1109/ROBOT.1994.350977.

[3] A. Delgado, C. Kambhampati, and K. Warwick. "Dynamic recurrent neural network for system identification and control". In: *IEE Proceedings - Control Theory and Applications* 142.4 (1995), pp. 307–314. ISSN: 1350-2379. DOI: 10.1049/ip-cta:19951873.

[4] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html.

[5] Martin T. Hagan, Howard B. Demuth, and Orlando De Jesús. "An introduction to the use of neural networks in control systems". In: *International Journal of Robust and Nonlinear Control* 12.11 (), pp. 959–985. DOI: 10.1002/rnc.727. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rnc.727. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rnc.727.

[6] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359 –366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: http://www.sciencedirect.com/science/article/pii/0893608089900208.

[7] K. J. Hunt et al. "Neural Networks for Control Systems: A Survey". In: *Automatica* 28.6 (Nov. 1992), pp. 1083–1112. ISSN: 0005-1098. DOI: 10.1016/0005-1098(92)90053-I. URL: http://dx.doi.org/10.1016/0005-1098(92)90053-I.

[8] Andrej Karpathy. *Convolutional Neural Networks (CNNs / ConvNets)*. http://cs231n.github.io/convolutional-networks/. URL: http://cs231n.github.io/convolutional-networks/.

[9] Duško Katić and Miomir Vukobratović. "Survey of Intelligent Control Techniques for Humanoid Robots". In: *Journal of Intelligent and Robotic Systems* 37.2 (2003), pp. 117–141. ISSN: 1573-0409. DOI: 10.1023/A:1024172417914. URL: https://doi.org/10.1023/A:1024172417914.

[10] Guy Katz et al. "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks". In: *CoRR* abs/1702.01135 (2017). arXiv: 1702.01135. URL: http://arxiv.org/abs/1702.01135.

[11] T. Kohonen. "The Self-Organizing Map". In: *Proceedings of the IEEE* 78.9 (1990), pp. 1464–1480. ISSN: 0018-9219. DOI: 10.1109/5.58325.

[12] Francesco Leofante et al. "Automated Verification of Neural Networks: Advances, Challenges and Perspectives". In: *CoRR* abs/1805.09938 (2018). arXiv: 1805.09938. URL: http://arxiv.org/abs/1805.09938.

[13] Zachary Chase Lipton. "A Critical Review of Recurrent Neural Networks for Sequence Learning". In: *CoRR* abs/1506.00019 (2015). arXiv: 1506.00019. URL: http://arxiv.org/abs/1506.00019.

[14] K. S. Narendra and K. Parthasarathy. "Identification and control of dynamical systems using neural networks". In: *IEEE Transactions on Neural Networks* 1.1 (1990), pp. 4–27. ISSN: 1045-9227. DOI: 10.1109/72.80202.

[15] Hasim Sak, Andrew W. Senior, and Françoise Beaufays. "Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition". In: *CoRR* abs/1402.1128 (2014). arXiv: 1402.1128. URL: http://arxiv.org/abs/1402.1128.

[16] J. Schroder et al. "Dynamic pneumatic actuator model for a model-based torque controller". In: *Proceedings 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation. Computational Intelligence in Robotics and Automation for the New Millennium (Cat. No.03EX694)*. Vol. 1. 2003, 342–347 vol.1. DOI: 10.1109/CIRA.2003.1222113.

[17] Adam P. Trischler and Gabriele M. T. D'Eleuterio. "Synthesis of recurrent neural networks for dynamical system simulation". In: *CoRR* abs/1512.05702 (2015). arXiv: 1512.05702. URL: http://arxiv.org/abs/1512.05702.

[18] S. Elanayar V.T. and Y. C. Shin. "Radial basis function neural network for approximation and estimation of nonlinear stochastic dynamic systems". In: *IEEE Transactions on Neural Networks* 5.4 (1994), pp. 594–603. ISSN: 1045-9227. DOI: 10.1109/72.298229.

[19] Weiming Xiang and Taylor T. Johnson. "Reachability Analysis and Safety Verification for Neural Network Control Systems". In: *CoRR* abs/1805.09944 (2018). arXiv: 1805.09944. URL: http://arxiv.org/abs/1805.09944.

[20] Weiming Xiang et al. "Verification for Machine Learning, Autonomy, and Neural Networks Survey". In: *CoRR* abs/1810.01989 (2018). arXiv: 1810.01989. URL: https://arxiv.org/abs/1810.01989.

Figures and Experiments

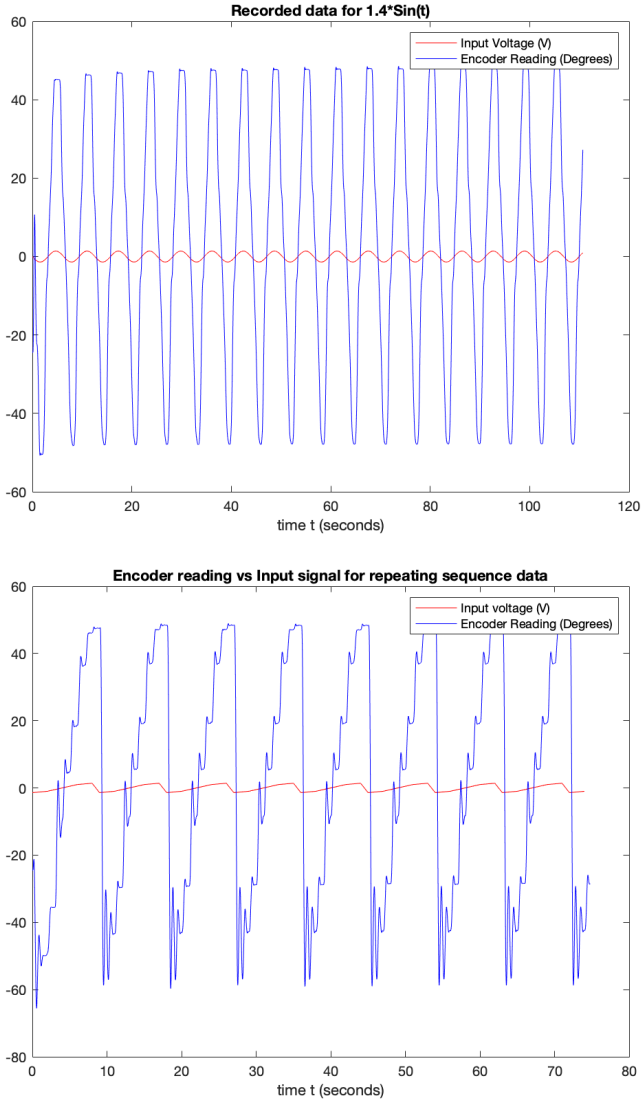

Figure A.2: Plot of the training data used to train the neural network

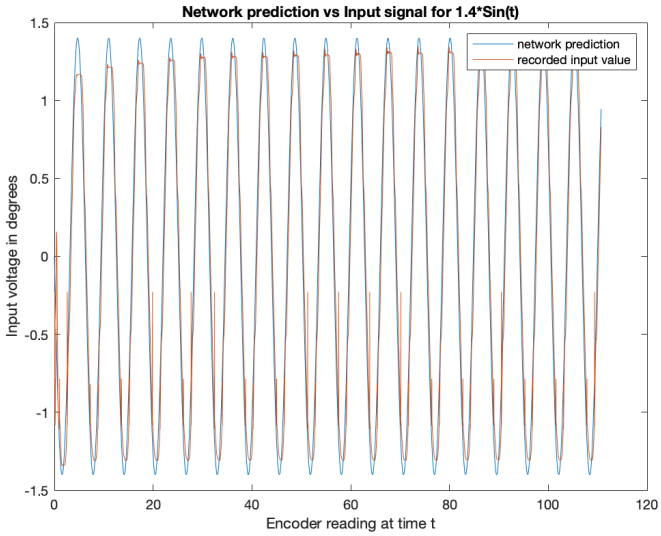

Figure A.3: Plot of the network's performance on the training data

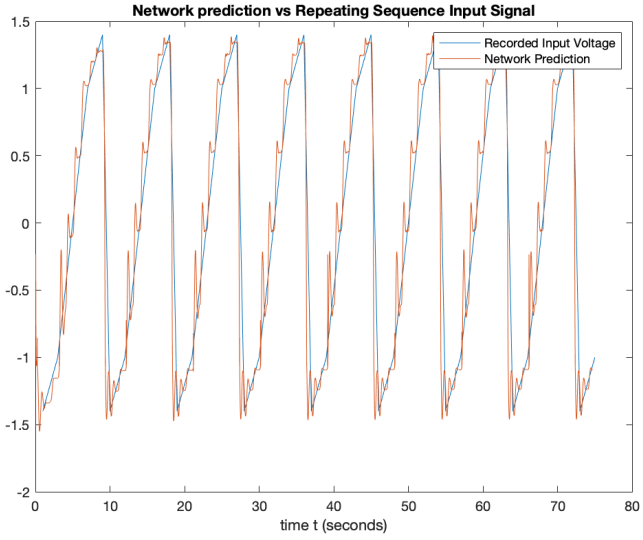

Figure A.4: Plot of the network's performance on separately generated test data. This data consisted of a sequence of encoder readings generated by the repeating sequence block in Simulink.

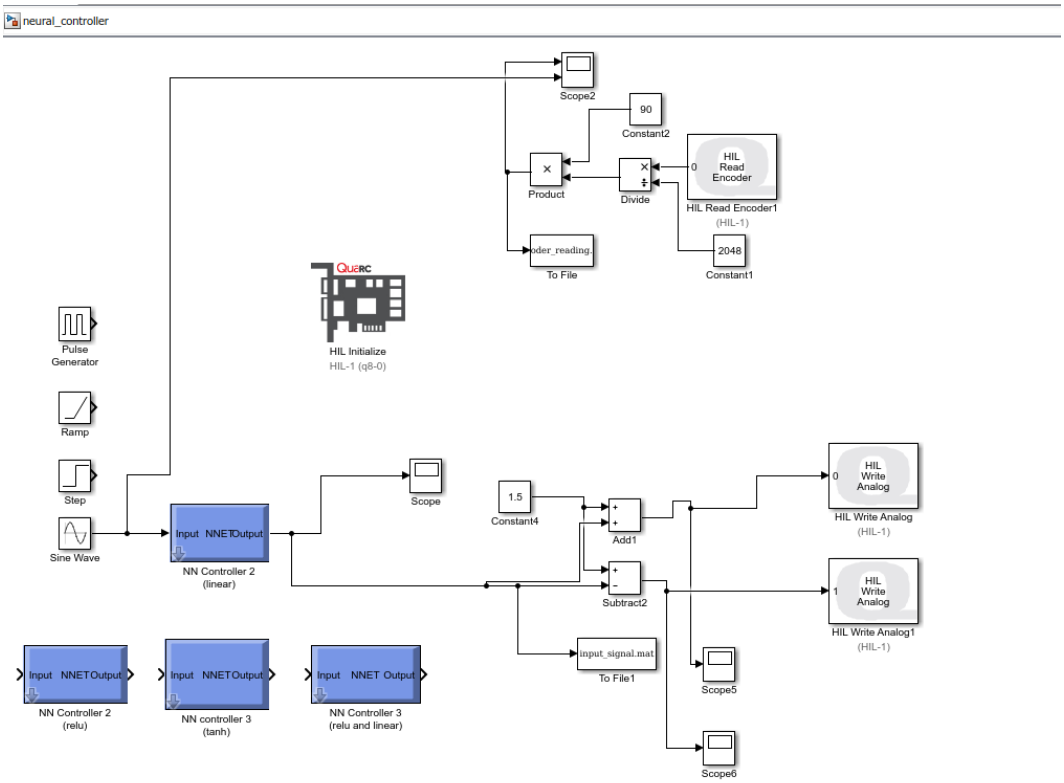

Figure A.5: Simulink representation of Figure 2.1. The network was able to correctly map desired angles to input voltages.

Figure A.6: Mechanical arm during simulations with input voltages produced by the neural network model