Intelligent Control of a Pneumatic Actuator Controlled Mechanical Arm

Fall 2018

I Introduction

In this lab, we seek to exploit the nonlinear functional mapping competence of neural networks in order to control a pneumatically driven mechanical arm [5]. The pneumatic actuators, called *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 [1]. These muscles consist of an internal bladder surrounded by a braided shell that is attached to fittings on either end [1]. 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 according to its volume increase [1]. This shortening produces tension that can be coupled to a mechanical arm in order to induce rotation [5]. A diagram of a pneumatic actuated mechanical arm can be seen in Figure 1.

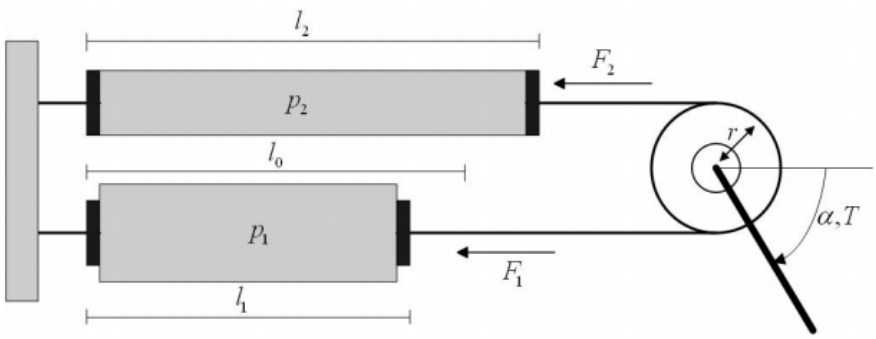

Figure 1: Diagram of Mckibben Actuated Mechanical Arm as shown in [5]

II Context and Origins

In recent years, Artificial Neural Networks have demonstrated an impressive ability to be used as frameworks for solving complex problems in numerous application domains [6]. In fact, the success of these models in contexts such as adaptive control, non-linear system identification [4], 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 due to their highly

parallel structure that allows for distributed processing, an ability to model complex non-linear systems [3], a capacity to handle large multivariate systems, and lastly their capability of being encoded efficiently within hardware [2]. However, the chief profit in making use of neural network models, is their ability to be used in synthesizing non-linear controllers for complex systems [2].

III Objective

In this lab, Students will learn how 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*. Students will gather sensor data from a voltage encoder so that to the network learns to map a desired angle to the specific voltage that will induce this orientation on the mechanical arm. The overall goal of the experiments is to demonstrate to Students how neural networks are trained and designed and further demonstrate their utility within control systems.

IV Pre-Lab Tasks

1. Read the following articles for an introduction to artificial neural networks. You will need to read these articles in order to answer the questions at the end of this lab.
   - A gentle introduction to neural networks: (https://towardsdatascience.com/a-gentle-introduction-to-neural-networks-series-part-1-2b90b87795bc
   - Stanford's Introudction to Neural Networks: (http://cs231n.github.io/neural-networks-1)
   - *Optional*: There are two more lectures offered by Stanford that can be found at the websites listed below. Feel free to skim those lectures as well.
     - http://cs231n.github.io/neural-networks-2/
     - http://cs231n.github.io/neural-networks-3/
     - http://cs231n.github.io/neural-networks-case-study/
2. Study the Rubbertator Kit Series in the lab and investigate the characteristics of Mckibben Actuators.
   - Why are two actuators needed to create a rotating force?
   - The Rubberrator experiences a non-linearity called hysteresis. Describe what hysteresis is.

Hint: The answers to the above questions can be found in the papers included in the references of this lab. Feel free to use Google as well.

V Experimental Setup

The mechanical arm consists of two Rubberators connected to a hinge joint. The angular position of the joint, in degrees, is 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 defined to be $2.5V$. To control the mechanical arm, one must alter the ratio of the maximum pressure given by

$$\frac{\text{Current Voltage}}{\text{Maximum Voltage}} * \text{Pressure input} \tag{1}$$

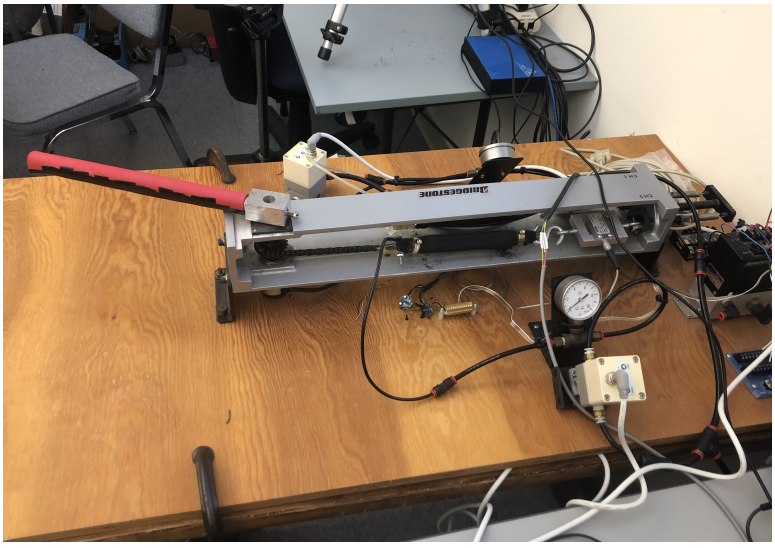

Figure 2: Mechanical Arm Experimental Setup

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 pressures 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. The aforementioned system has a unique feedback sensor which is an encoder that estimates the angle of the mechanical arm's orientation in degrees.

VI Lab

Before we can train a neural network, we must collect a sufficient amount of data so that the network can learn a function that correctly maps the angle inputs to the corresponding voltage values that generate this orientation. The first part of this lab will involve collecting data by simulating the system using a variety of voltage input signals (ramp, step, sinusoidal) and recording the encoder values during each simulation. Once the data has been collected, the MATLAB ® neural network

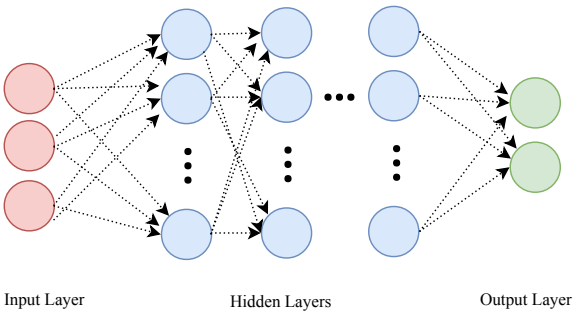

Figure 3: Simple Feed-forward Architecture

toolbox will be used to train neural network of suitable performance. The lab allows students to investigate a wide variety of activation functions and network architectures so that they can learn how to optimize the parameters of a neural network in order to maximize its prediction accuracy.

VII Procedure

1. Study the experimental setup in order to understand how the rubberators induce rotation in the mechanical arm. Identify the voltage encoder and reason about the feedback control system in its totality.
2. Log in to the computer using your VUNetID and Password.
3. Ensure that the power supply located to the right of the experimental work bench has been switched on, and additionally that the Quarc board power supply switch has also been turned on.
4. Next, turn the topmost valve clockwise of the air tank located to the right of the workbench until the **pressure reads 40psi**. You should really only have to rotate the valve a little.
5. Open Matlab version **2017a**. Failure to do so will result in numerous errors.
6. On the desktop, navigate to the Fuzzy Control Team Files folder and double-click on it.
7. Open the *NeuralControlFall2018* directory.
8. Right-click on the *neural_controller.slx* file and select *open with Matlab 2017a*
9. Next, right-click on the *NeuralTraining.mlx* file and select *open with Matlab 2017a*. You will use this file later in the lab. (This file is a Matlab live script)

I. Data Collection

To train the neural network we must collect a sufficient amount of data so that it can learn the relationship between the voltage inputs and the encoder's estimation of the mechanical arm's orientation.

1. Before we begin let us first simulate the system so that you can get a sense of how the system operates.

(a) *Click Quarc* → *Build* on the Matlab toolbar at the top of the screen to build the code needed to simulate the system. This will take some time to complete.

(b) Once the build process is over, click on *Simulation* → *Start Real-Time* to run the model file. Alternatively click on the bright green play button on the tool bar. The mechanical arm should begin to move in response to a sinusoidal input signal as show in Figure 4. **Note: if Matlab displays any errors and the system does not simulate properly, please refer to Section X**.

2. When the simulation is running, the Simulink model will be recording readings of the voltage encoder's estimation of the mechanical arm's orientation as well as the input voltage signals that induce this position. These recordings are stored in the *NeuralControlFall2018* directory as *encoder_reading.mat* and *input_signal.mat* respectively. To train a neural network to model this relationship, we must collect data using a variety of input signals.

3. Alter the Simulink diagram so that the bottom part of the model looks like Figure 5. Double-click on the Sine Wave Block and set the amplitude to 1.4. Build the model as before, and simulate the system for 100 seconds. Rename the corresponding files that are created in the *NeuralControlFall2018* folder as *encoder_reading_sin1.4.mat* and *input_signalsin1.4.mat* respectively.

4. Repeat this step for the other input signals present within the provided Simulink model and label the output files with the appropriate input signal name. (i.e *input_signalpulsegenerator.mat*).

Checkpoint Question: Which of the input signals do you think will be most useful in training the neural network? Explain.

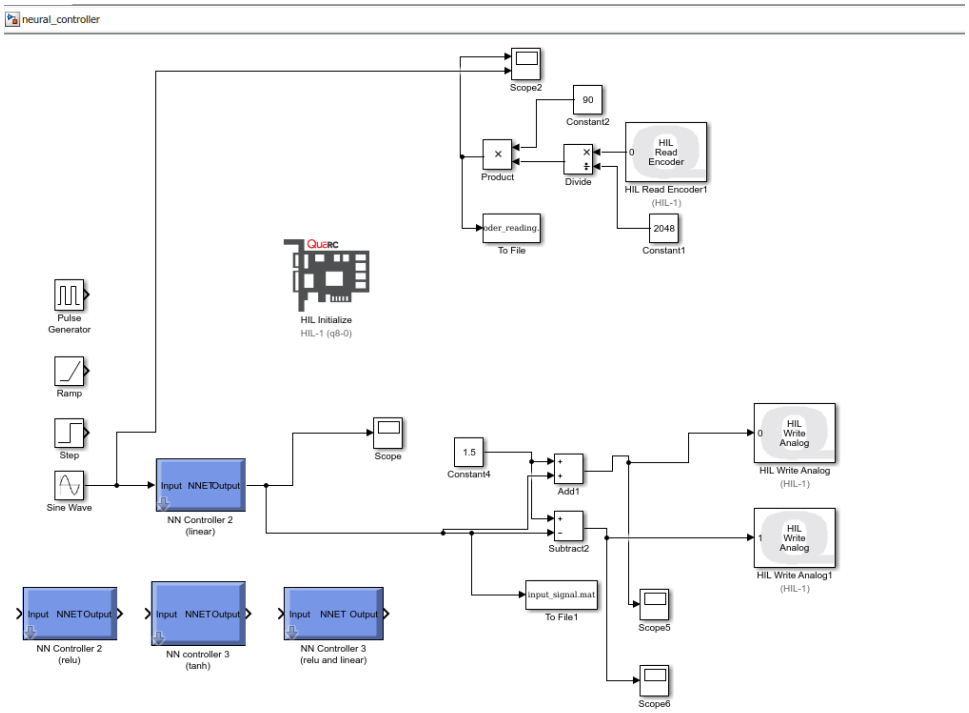

Figure 4: Simulink Model for interacting with the mechanical arm

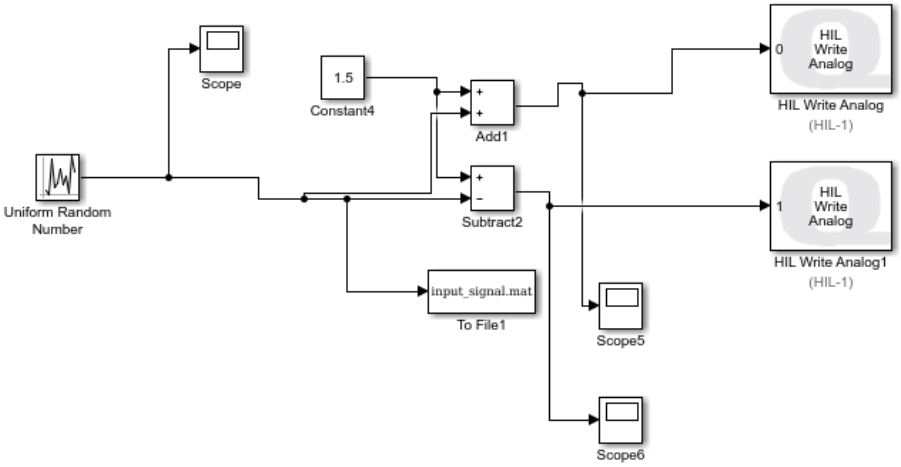

Figure 5: Data Collection Model

II. Network Training

At this point we have collected enough data to train a neural network model and in this lab we will train the network using the sinusoidal input signal. However if you wish to train the network with the other data that we collected this can be done simply by altering step 1 and step two in the instructions that follow.

1. Open *NeuralTraining.mlx* file. This file contains Matlab code to create a neural network model. We have included comments so that you can understand what is going on at each step. It may be useful to briefly skim the code before continuing on in the lab.

2. The first line in the Matlab script loads the input data which is located in the file specified by "input_signal_sin1.4.mat". However if you used a different naming convention, change this string to the name of the file name that contains sinusoidal input data.

3. Load the encoder recordings data in a similar fashion to step one. (i.e "encoder_reading_-sin1.4.mat")

4. The data that is returned by the above statements is structured into a two dimensional list of vectors. The first vector stores the time instants where the input signal data and encoder readings were sampled, and the second vector stores the readings. Thus, the lines following the load statements select only the readings as this is the most pertinent information for the neural network.

5. Unfortunately as is sometimes the case with data collection devices, **The encoder's readings are off by 24.5 degrees**. Thus the data that we recorded does not truly exemplify the behavior of the mechanical arm. In order to rectify this, we shift the encoder readings by -24.5 in order to account for this error.

6. Next, we define our neural network model. The line *net=feedforwardnet([10 10 10], trainbr)*, defined a feed-forward neural network with three layers that consist of 10 neurons. Change these values to [5 5 5]. The second parameter 'trainbr' specifies the training algorithm used to tune the network's weights and in this lab we will use Batch Gradient Descent with Bayesian regularization. There are numerous other algorithms and if you are curious I have attached some relevant papers in the references.

7. The next several lines define the activation functions for each layer and we have commented the code indicating what these function names correspond to. Change layer 1 and layer 2's activation functions to the ReLU function and define the output layer's function to be a linear activation function (purelin).

8. Next, we define the size of the input and output layers of the network and specify the initialization functions. Do not worry too much about these functions for now. However for the curious student, the way you initialize the network's weights before training has a significant influence on the performance of the training algorithms. In our case there is only one input, the encoder angle reading, and the output will be the voltage required to induce this angle.

9. Once we have defined the neural network, and have loaded the training data. The next step is to train the network. The remaining lines in the file trains the network that we have

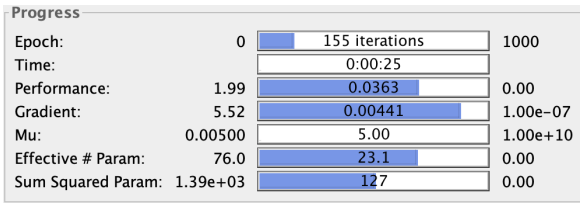

Figure 6: Training Iterations. Once you notice that the performance of the neural network has stopped improving, you can stop the learning process by clicking stop training.

defined. Additionally we plot the performance of the network and calculate the network's mean squared error performance. Intuitively the mean squared error performance of the system is a measure of how well the neural network produces correct predictions.

10. Once the network has finished training we plot the network's performance on the training data. However this a naive gauge of the networks ability. To truly test our network, we must see how it performs on unseen data.

11. In the following lines we show you how to test the network using some data we collected using Simulink's repeating sequence block. Evaluate the networks performance using the signal pulse generator data and attach those plots to this lab.

12. Lastly, we generate a Simulink block so that you can gauge the performance of your network on the mechanical arm. To do this simply type gensim(net) in the command window. This will generate a Simulink block that contains the structure of the network in another file. Drag the network into the *neural_controller.slx* Simulink model and connect it as in Figure 4. Change the Sinusoidal input amplitude from 1.4 to 45, so that it now represents a desired angle. Build the model as before and start the simulation.

13. The network should perform well on this input data. If all goes well, you have just successfully created a neural network to estimate the correct voltage input for a user's desired input. Well done! If you have extra time feel free to test the other networks that we have included in the file. Although the network that we used had 15 neurons, this task is not overly complex so it can be done with only 4 neurons.

VIII Before you leave...

After you have finished and are ready to leave, please make sure to **turn the tank valve off by rotating the valve counterclockwise!** Otherwise the tank will continue to lose air and no one else will be able to run the lab in the future.

IX Post-Lab Questions

1. What is an activation function and how is it related to the neural network's performance?

2. How much data is needed to achieve a suitable performance in the network?

3. Discuss the process of collecting data using a camera instead of a voltage encoder? What will be different in the neural network training process? What will be different about the network architecture? *Hint: briefly survey the following article* http://cs231n.github.io/convolutional-networks/

4. Since a network with only four neurons is sufficient to learn a relationship between the encoder readings, do you think that training a neural network is worthwhile? Explain.

5. Search the internet for one real world application of a neural network in a controls setting, and provide a brief summary below. **Provide a suitable citation.**

X Troubleshooting

Error. *Error occurred while executing External Mode MEX−file 'quarc_comm': An operating system specific kernel-level driver for the specified card could not be found. The card or driver may not be installed...*

To rectify this error perform the following steps:

1. The error is caused by the Windows operating system not being able to find a set of specific device drivers required to use the Quarc board. Thus we must install some software onto the computer. In order to do this you must have administrative access on the computer. If you do not have administrative access please contact Aaron Cooper. His email is aaron.r.cooper@vanderbilt.edu

2. Ensure that the NI-DAQmx drivers are installed. They can be found at `www.ni.com/drivers/`. **Make sure to install the NI-DAQmx drivers only**. Once you have installed the drivers please restart the computer in order for the changes to take effect.

3. If you are still experiencing issues, then there is a problem with the Quarc installation. To repair the Quarc installation:

   - Navigate to the *Local disk* in the file explorer
   - Open the *temp* directory
   - Inside the temp directory there should be a directory named 2.6. If this directory does not exist please contact Aaron copper to obtain the quarc installation software.
   - Inside the *2.6* directory there should be an executable named *Install quarc*
   - Run the *Install quarc* executable and click **repair** once it opens up.

- The repair installation will take several minutes and at the end of installation the installer will display an error related to the Quarc software license. **disregard this error**. Close the installer and restart the computer.
- *Warning:* You may have to perform the re-installation of Quarc every time you use the system in the future, as it appears that these settings are not retained when the computer is shut down.

References

[1] Chou, C.-P., and Hannaford, B. Static and dynamic characteristics of mckibben pneumatic artificial muscles. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation* (May 1994), pp. 281–286 vol.1.

[2] Hunt, K. J., Sbarbaro, D., Żbikowski, R., and Gawthrop, P. J. Neural networks for control systems: A survey. *Automatica 28*, 6 (Nov. 1992), 1083–1112.

[3] Katić, D., and Vukobratović, M. Survey of intelligent control techniques for humanoid robots. *Journal of Intelligent and Robotic Systems 37*, 2 (Jun 2003), 117–141.

[4] Narendra, K. S., and Parthasarathy, K. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks 1*, 1 (March 1990), 4–27.

[5] Schroder, J., Erol, D., Kawamura, K., and Dillman, R. 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)* (July 2003), vol. 1, pp. 342–347 vol.1.

[6] Xiang, W., Musau, P., Wild, A. A., Lopez, D. M., Hamilton, N., Yang, X., Rosenfeld, J., and Johnson, T. T. Verification for Machine Learning, Autonomy, and Neural Networks Survey. *CoRR abs/1810.01989* (2018).