Spring 7-16-2018

Emulating Balance Control Observed in Human Test Subjects with a Neural Network

Wade William Hilts
Portland State University

Recommended Citation
Hilts, Wade William, "Emulating Balance Control Observed in Human Test Subjects with a Neural Network" (2018). *Dissertations and Theses.* Paper 4499.
https://doi.org/10.15760/etd.6383

Emulating Balance Control Observed in Human Test Subjects with a Neural Network

by

Wade William Hilts

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Mechanical Engineering

Thesis Committee:
Alexander J. Hunt, Chair
David Turcic
Faryar Etesami

Portland State University
2018

Abstract

Human balance control is a complex feedback system that must be adaptable and robust in an infinitely varying external environment. It is probable that there are many concurrent control loops occurring in the central nervous system that achieve stability for a variety of postural perturbations. Though many engineering models of human balance control have been tested, no models of how these controllers might operate within the nervous system have yet been developed. We have focused on building a model of a proprioceptive feedback loop with simulated neurons. The proprioceptive referenced portion of human balance control has been successfully modeled by a PD controller with a time delay and output torque positive feedback. For this model, angular position is measured at the ankle and corrective torque is applied about the joint to maintain a vertical orientation. In this paper, we construct a neural network that performs addition, subtraction, multiplication, differentiation and signal filtering to demonstrate that a simulated biological neural system based off of the engineering control model is capable of matching human test subject dynamics.

Acknowledgements

I would like to express my gratitude to my professors and advisor at Portland State for guiding me through all of the obstacles I faced during my research. Dr. Alex Hunt gave me the inspiration to pursue a bio-inspired robotic control system, and taught me a lot about how to frame a research question and to express my hypothesis and results in a coherent and concise manner. Dr. David Turcic provided a wealth of experiential knowledge in the classical controls regime, showed me how to connect control theory and practice, and all the complications that arise with real-world systems. Outside of PSU, I received advice and guidance for Robert J. Peterka (OHSU), who provided me with the foundational experimental data and counsel on my approach to human balance modeling. Nicholas Szczecinski developed the neural subnetworks that were the building blocks of the controller I designed, and provided immense technical support and counseling on the subject. Additionally, I'd like to acknowledge my lab partners Alex Steele, Tiffany Hamstreet and Karmand Rasheed for providing me with good company, advice and insight on my research that was based off of their unique perspectives and experiences.

Contents

List of Figures

Nomenclature

G_p	The plant model transfer function, represented in the Laplace domain
G_c	The controller transfer function, represented in the Laplace domain
H_T	The controller in the feedback loop, represented in the Laplace domain
τ_d	The Padé approximant, represented in the Laplace domain
K_p	Proportional gain coefficient
K_d	Derivative gain coefficient
K_t	Positive torque feedback gain coefficient
ω_c	Low-pass filter cutoff frequency, rad/s
PD	Short for Proportional-Derivative
J	Moment of inertia, kg·m^2/s^2
b	Damping coefficient
θ	Ankle joint angle, rad
m	Mass, kg
g	Acceleration due to gravity, m/s^2
h	Center of mass height, m
C_m	Membrane capacitance, F
G_m	Membrane conductance, S
V	Absolute membrane potential, V
U	Membrane potential difference from neuron resting potential, V
I	Current, A
ΔE	Operating range of the synaptic connection between neurons, V

Chapter 1

Introduction and Motivation

In recent years, neuromorphic computing chips with promises to revolutionize computing technology have become available. These chips effectively model neurons and synapses in a compact architecture that consumes orders of magnitude less power than a comparable digital system [10, 14]. However, there are few synthetic neural control algorithms that can utilize these chips. Most synthetic neural research has been focused around pattern recognition, image processing, or decision making [3, 5, 6]. Almost all of these systems do not require quick reactions to external changes or interaction with an unpredictable environment.

Several synthetic neural control systems have been developed with varying success. A few of them are used to control legged systems, and must quickly process sensory data and make control actions to maintain effective interaction with the surrounding environment [7–9]. However, these neural systems are built on individual case studies and though insights can be gathered from how the control systems worked, they are not easily portable to new problems or systems.

To this end, tools that assist in creating neural controllers for new systems have been developed. Nengo provides methods to set up spiking neural systems and then train them to produce specific desired outputs [2]. I have crafted methods in which parameters in small neural systems can be set analytically to perform mathematical operations such as addition, subtraction, multiplication, division, differentiation, and integration [17]. These different subnetworks can be developed independently and then added together to perform complex mathematical operations. I have also developed tools for analyzing and setting parameters in pattern generating circuits common in locomotion [16]. However, it is unclear how effective these methods are when applied to even more dynamic and unsteady control problems encountered in the robotics world.

It is my hypothesis that the analytical methods I have developed, combined with classical control techniques will provide a reasonable starting point for developing dynamic controllers. I test this hypothesis by developing a neural controller that is analogous to a classical control model fit to human test subjects [12, 13]. In this model, a PD controller with time delay and low-passed positive feedback uses corrective torque to keep an inverted pendulum system upright. I first use the engineering model to determine the proportional, derivative, time delay, filter and positive feedback parameters that match the stability characteristics of the control system. Then, I employ neural network design methods [17] to lay out the framework and set the parameters of the neural network, the resulting neural controller is shown to match the classical control system behavior.

Chapter 2

Classical Control Methods

2.1 Methods for System Identification and Controller Design

2.1.1 A Linear Model for Human Balance

The linear model for human balance control in this paper was based off a model proposed by Peterka derived from human test subject data collected on a tilting platform [12]. The test subjects in Peterka's experiment had profound vestibular loss, and the data was collected with their eyes closed. Additionally, the subjects were strapped to a fixture that allowed them to only use corrective torque at the ankle joint. Human balance control is postulated to rely on vestibular, proprioceptive and visual feedback [1, 13], this experiment effectively eliminated the contribution of vestibular and visual feedback while constraining the corrective output to only torque at the ankle joint.

Frequency response data points were collected for these test subjects and Peterka proposed a simple control architecture that fit the test data. Peterka's plant model for the human body consisted of a simple inverted pendulum model, free of any damping effects. He also proposed a model for the control response that includes a time delay, positive force feedback with a low-pass filter, a PD controller in the standard controller position and a feedforward controller modeling the passive muscle dynamics (unaffected by the time delay) [12]. Peterka's results provide an engineering control model that has been tuned to match human test data in simulation on an inverted pendulum plant model. This engineering model was used as the basis for the neural control structure in this paper.

2.1.2 Identifying the Plant Model Used in the Experiment

The first step to implementing a control system that matches human balance characteristics is accurately identifying the plant's dynamic response to inputs, and fitting a model to this data. The system being controlled was a single-jointed inverted pendulum model with a motor placed at the base joint (Figure 2.1) modeled by the time domain differential equation:

$$J\ddot{\theta} + b\dot{\theta} - mgh \cdot sin(\theta) = T_c. \tag{2.1}$$

Where θ is the angular position, T is the commanded torque, J is the moment of inertia, b is the damping ratio and the mgh term is the destabilizing torque due to gravity. We assume an ideal motor that produces the commanded torque instantly, the HEBI X8-9 torque control motor is capable of behaving as such according to manufacturer specifications. We also use the small-angle approximation, $sin(\theta) = \theta$, to linearize the model.

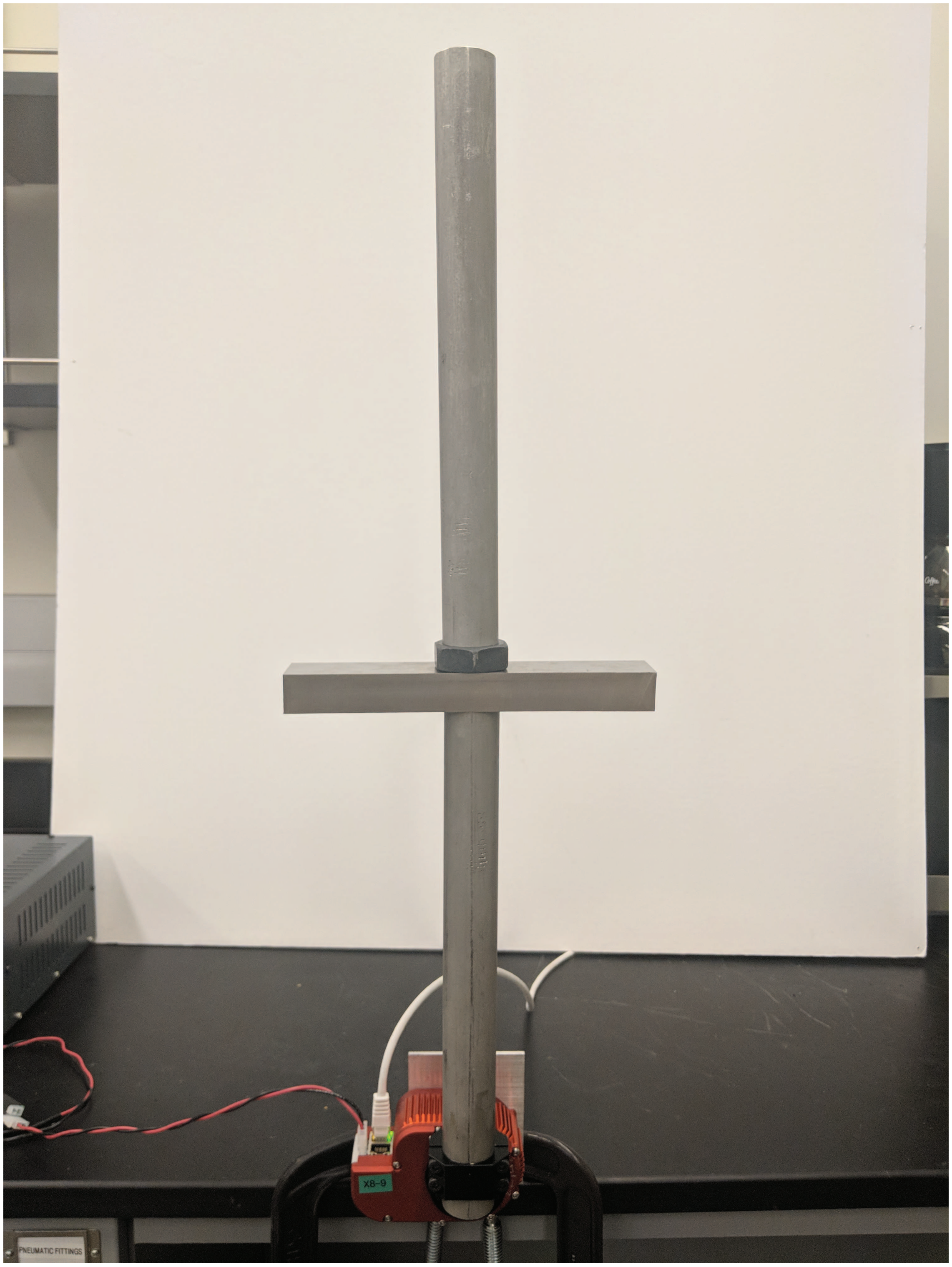


Figure 2.1: Controlled system in this experiment. The system is comprised of a several pieces of steel rigidly fastened together, with torque controlled motor acting as the base joint.

2.1.3 Plant Model Validation

System identification is performed using a closed loop controller because the inverted pendulum plant model is unstable for open-loop position control. While certain parameters of the system can be measured or calculated individually (e.g. moment of inertia, mass), the damping ratio of the system must be determined empirically. A proportional controller was used to experimentally determine the gain and phase shift of the system output with a closed loop transfer function in the Laplace Domain:

$$\frac{\theta_{act}}{\theta_{des}} = \frac{K_p}{Js^2 + bs - mgh + K_p} = \frac{G_c G_p}{1 + G_c G_p} \tag{2.2}$$

Where $G_c = K_p$ represents the proportional controller and G_p is the plant model. An 8-degree peak to peak sinusoidal commanded position signal was used. The proportional gain, K_p, was set to 30 and the moment of inertia, mass and center of mass height were measured before the experiment. These values were found to be $J = 0.44$ kg·m^2, and $mgh = 9.5$ kg·m^2/s^2.

A state space model of this system in MATLAB was constructed, and the theoretical form of the model enforced by using the 'greyest' linear function fitting tool. This was done by setting the damping ratio parameter completely free and fixing the other known parameters, as the motor used to control the system had more complex dynamics at torque values near zero and introduced unknown damping to the system. The greyest function was used to optimize the damping ratio value to closest match experimental results, by minimizing the error between the model prediction and the experimental data (Appendix A.6 and A.5).

2.1.4 Designing a Controller that Will Produce a Closed Loop Response Similar to the Test Subject Data

After identifying a linear plant model for the inverted pendulum system, I developed a controller that produces similar frequency response characteristics as the human test subjects. The proposed control system takes a single input, the inverted pendulum's angular position, and outputs a corrective torque that is applied at the base joint. This control system can be represented by the block diagram in Figure 2.2 and closed loop transfer function, Equation 2.3:

$$\frac{\theta_{act}}{\theta_{des}} = \frac{\tau_d G_c G_p}{1 - \tau_d G_c H_T + \tau_d G_c G_p}, \text{ where} \tag{2.3}$$

$$G_p = \frac{1}{Js^2 + bs - mgh} \tag{2.4}$$

$$G_c = K_p + K_d s \tag{2.5}$$

$$H_T = \frac{K_t \omega_c}{s + \omega_c} \tag{2.6}$$

$$\tau_d = \frac{(-\tau s + 2)}{(\tau s + 2)} \approx e^{-\tau s} \tag{2.7}$$

where K_p is proportional gain, K_d is derivative gain, K_t is positive torque feedback gain, ω_c is low-pass filter cutoff frequency and τ is a time delay. A first order Padé approximant was used to linearize the time delay 2.7.

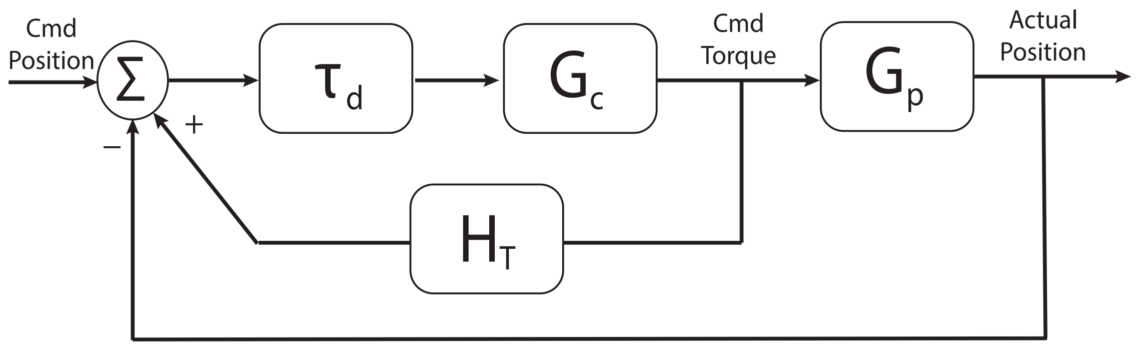


Figure 2.2: Block diagram of human balance control engineering model. There are two nested feedback loops here. The inner feedback loop consists of a time delayed controller receiving a low-passed positive torque feedback signal. The outer loop is a negative angular position feedback loop, drawn from the output of the plant model's (G_p) response to the torque signal.

2.1.5 Control Model Validation and Signal Processing

Using a similar method as in the plant model identification, the above transfer function can be converted into state space form and plugged into the MATLAB greyest function (Appendix A.4). We set the time delay, low-pass filter cutoff frequency, and the proportional, derivative and torque positive feedback gains as free parameters and used the greyest function to minimize the error bet en the test subject data and the controlled system's predicted closed loop frequency response. This process produced a controller for my inverted pendulum plant model that would emulate Peterka's model controller for the response of a blindfolded human with vestibular loss on a tilting platform using only corrective torque at the ankle joints.

As with many control system applications, high frequency noise wreaks havoc on the derivative component of a controller, and frequencies higher than the Nyquist frequency must be filtered out. The control loop was run at a frequency of 150 Hz. We included two second order Butterworth low-pass digital filters, each with cutoff fre-

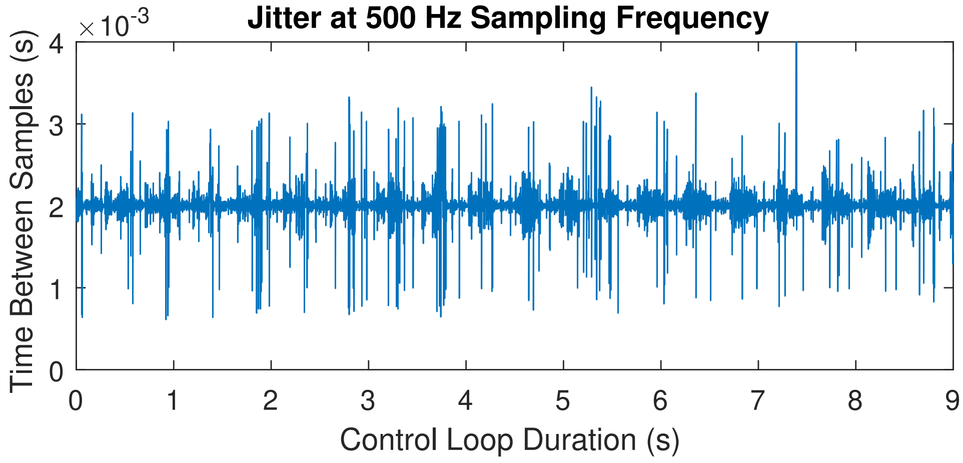


Figure 2.3: Jitter observed at 500 Hz sampling frequency. The sampling period varies by approximately 50 percent of its mean value.

quencies of 50 Hz, to filter the incoming position feedback and outgoing torque command signals. Additionally, the Windows 10 operating system that MATLAB is running on introduces a phenomena known as jitter, introducing a variance in the timing of the control loop period. For this system, the jitter has a peak magnitude of about one millisecond, independent of the sampling frequency.

The controller was validated using MATLAB's system identification toolbox to derive the closed loop transfer function of the controlled system. We used time domain experimental data to obtain a transfer function rather than fitting to frequency domain data. Filtered Gaussian white noise was sent as an input, with a low-pass filter applied that removed frequencies above 2 Hz. The max amplitude was set at 8 degrees peak to peak, which represented the majority of the operating space observed in the human data [12]. We took the input and output time domain data and fit it to a fourth order transfer function using the MATLAB System Identification toolbox based on the plant model and the controller design that was defined in Equation 2.3. The resulting closed

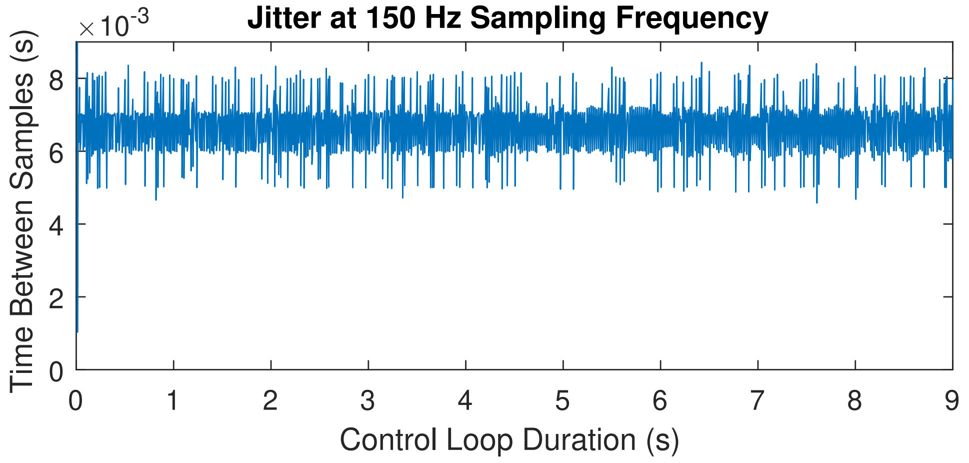


Figure 2.4: Jitter observed at a 150 Hz sampling frequency. The sampling period variance is approximately 20 percent of its mean value.

loop system should be well represented by a transfer function with 4 poles and 3 zeros. The transfer function fit to this data was then compared to the human data and theoretical prediction.

Chapter 3

Neural Network Control Methods

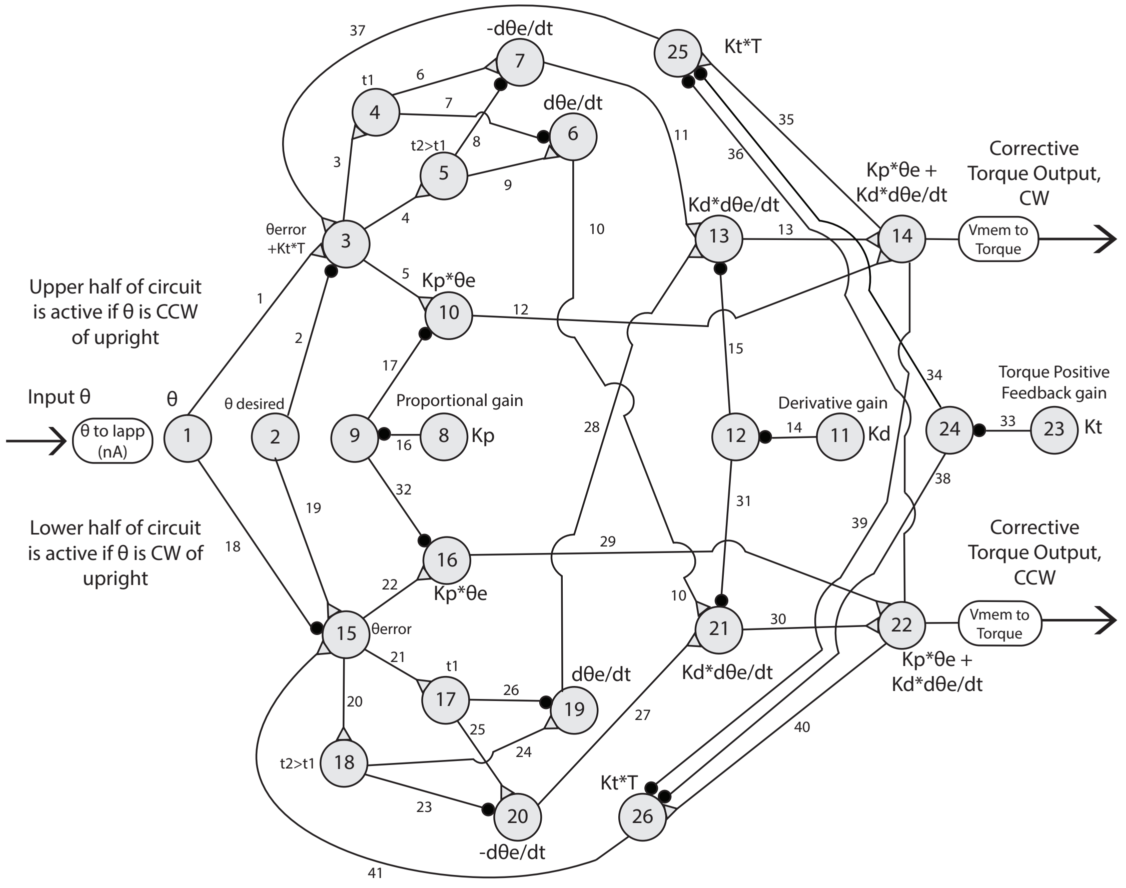
3.1 Neural Network Simulation

I created a neural network controller, shown in Figure 3.1, to emulate a PD controller with a time delay and low passed positive torque feedback. This was done by assembling a series of subnetworks with defined behaviors. The neurons and synapses in each subnetwork are assigned specific characteristics and connections to approximate the mathematical operations of the classical controller. This work utilizes a leaky, integrating, non-spiking neuron model. Information is encoded in the neurons' membrane voltage, and is transmitted via synaptic connections. The membrane voltage of a neuron is governed by:

$$C_m \frac{dV}{dt} = I_{leak} + I_{syn} + I_{app} \tag{3.1}$$

Where C_m is the membrane capacitance, V is membrane voltage, and I_x are the various current sources and sinks.

$$I_{leak} = G_m(E_r - V) \tag{3.2}$$



-dθe/dt
Kt*T
dθe/dt
Corrective Torque Output, CW
Kp*θe + Kd*dθe/dt
Kd*dθe/dt
θerror +Kt*T
Vmem to Torque
Kp*θe
Upper half of circuit is active if θ is CCW of upright
Proportional gain
Derivative gain
Torque Positive Feedback gain
Input θ
θ to Iapp (nA)
θ desired
Kp
Kd
Kt
Lower half of circuit is active if θ is CW of upright
Corrective Torque Output, CCW
θerror
t1
t2>t1

Figure 3.1: A network of neurons and synapses that outputs a torque command based on a joint angle input signal. The circuit is a collection of interconnected addition, subtraction, multiplication and derivative subnetworks and is broken into two sections, each governing the clockwise and counterclockwise regions (about the marginally stable midpoint) of the pendulum system. CW and CCW torque response signals are manifest in the membrane potentials of neurons 14 and 22.

Where G_m is the membrane conductance. Neurons transmit information via synapses. This current, I_{syn}, is defined as:

$$I_{syn} = \sum_{i=1}^{n} G_{s,i}(E_{s,i} - V). \tag{3.3}$$

Where $G_{s,i}$ represents the synapse conductance of the ith synapse. The synapse conductance can be described by a piecewise function:

$$G_{s,i} = \begin{cases} 0 & V_{pre} < E_{lo} \\ g_{s,i}\frac{V_{pre}-E_{lo}}{E_{hi}-Elo} & E_{lo} \leq V_{pre} \leq E_{hi}. \\ g_{s,i} & V_{pre} > E_{hi} \end{cases} \tag{3.4}$$

The above equation parametrizes the range over which postsynaptic neurons receive current from the presynaptic neurons. Below the value of E_{lo}, there is no current, between E_{lo} and E_{hi}, there is a linear increase in current until saturation at E_{hi}, in which the postsynaptic neuron receives no additional current when the presynaptic neuron's voltage goes higher.

I_{app}, is an external stimulus current. For the purpose of this simulation, the external stimulus current is injected into a neuron to represent outside information, such as the angular position of the inverted pendulum model, a descending command from the brain, or unmodeled feedback pathways.

To more easily describe neural arithmetic operations mathematically, I employ the simplifying definition:

$$\Delta E_{s,i} = E_{s,i} - E_{r,post} \tag{3.5}$$

Where $\Delta E(s,i)$ is the potential difference between the synaptic equilibrium potential and the postsynaptic neuron's resting potential.

3.1.1 Subnetworks

The complete neural controller is composed of fundamental building blocks, called subnetworks. Each subnetwork performs a specific operation on an input signal to produce a desired output. A graphical representation of each subnetwork is shown in Figure 3.2. Throughout this section, I will point to Figure 3.1 to provide examples of where the subnetworks are employed in the overall controller.

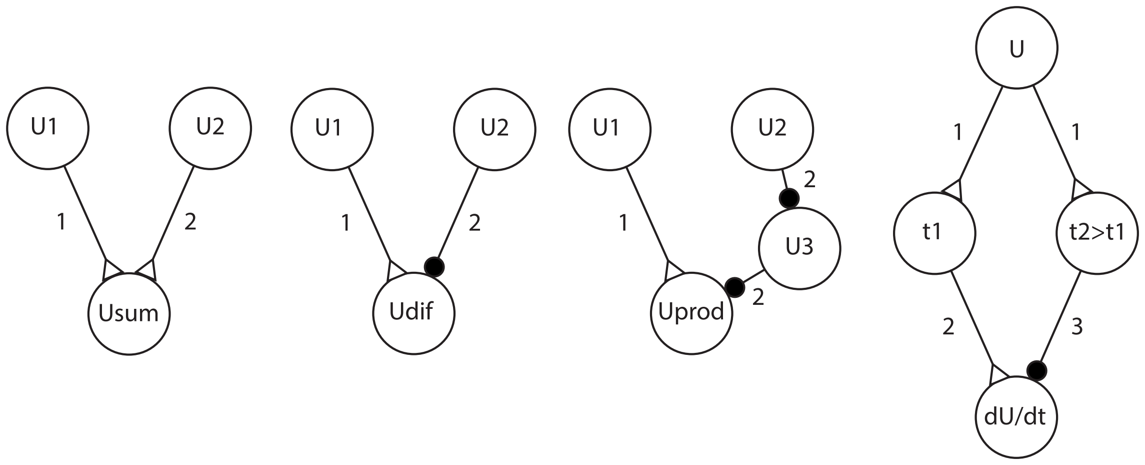


Figure 3.2: Graphical representations of the neurons and synapses. From left to right: addition, subtraction, multiplication and derivative subnetworks. Synapses terminating in a triangle are excitatory, whereas the shaded circular terminals are inhibitory synapses.

The following optimal parameters were calculated using Szczecinski's methods [16, 17] and are for a network designed with $R = 20$.

Neural Addition

Neurons can perform elementary addition, where two presynaptic neurons resting potentials can be summed and reflected in a postsynaptic neuron. For example, in the neural controller, the proportional (neuron 10) and derivative (neuron 13) contributions to the CW output torque (neuron 14) are all part of an addition subnetwork. Optimal parameters are:

$$g_{s,1} = g_{s,2} = 0.115\ \mu S \quad \Delta E_{s,1} = \Delta E_{s,2} = 194\ mV$$

a gain can be applied to the $U1$ and/or $U2$ input signals by multiplying $g_{s,1}$ or $g_{s,2}$ by a desired gain factor. This is useful because of the order of magnitude limitations that will be outlined in the multiplication network in Section 3.1.1.

Neural Subtraction

Subtraction is a very important calculation for closed loop control systems with negative feedback. For example, the input angle (neuron 1) and desired angle (neuron 2) are subtracted to produce a CCW error signal (neuron 3). The selected parameters are:

$$g_{s,1} = 0.115\ \mu S \quad g_{s,2} = 0.55775\ \mu S$$

$$\Delta E_{s,1} = 194\ mV \quad \Delta E_{s,2} = -40\ mV$$

, addition and subtraction networks can be used together in any number of combinations, allowing a single neuron to add and subtract multiple values at once.

Neural Multiplication

Multiplication is required to apply gain to a signal. This network can apply a gain of 0.1 to 1.0 and retain a relatively linear behavior across varying input signal magnitudes. Thus, in my control circuit it regulates the overall gain on a signal. Neurons 8, 9 and 10 are an example of a multiplication subnetwork that is applying a gain proportional to the CCW error signal fed into neuron 10. The stimulus current on the $U2$ neuron (see Figure 3.2) determines the gain value. The selected parameters are:

$$g_{s,1} = 0.115\ \mu S \quad g_{s,2} = 20\ \mu S$$

$$\Delta E_{s,1} = 194\ mV \quad \Delta E_{s,2} = -1\ mV$$

With tonic stimulus applied to $U3$: $I_{stim,U3} = 20\ nA$.

Neural Differentiation

A neural subnetwork can compute the derivative of an input neuron by routing the input neuron's membrane voltage to two separate neurons with different membrane capacitances. This difference in membrane capacitance creates a response time-delay between the neurons. Subtracting these neurons' membrane potentials gives an approximation similar to a backward difference function. In the neural controller, neurons 4 and 5 are time-shifted versions of the CCW error signal in neuron 3. They are subtracted to create a positive derivative signal (neural 6) and a negative derivative signal (neuron 7). The selected parameters are:

$$g_{s,1} = g_{s,2} = 0.115\ \mu S \quad g_{s,2} = 0.55775\ \mu S$$

$$\Delta E_{s,1} = \Delta E_{s,2} = 194\ mV \quad \Delta E_{s,3} = -40\ mV$$

$$\tau_1 = 0.1\ ms \quad \tau_2 = 8.0\ ms$$

3.1.2 Subnetwork Tuning

Each neural subnetwork must be tuned to match the gains and behavior specified in the classical controller design. This is achieved by isolating portions of the network, inputting test signals and observing the output. For example, the derivative gain subnetwork is tuned to match the specified gain of the classical controller, shown in Figure 3.3.

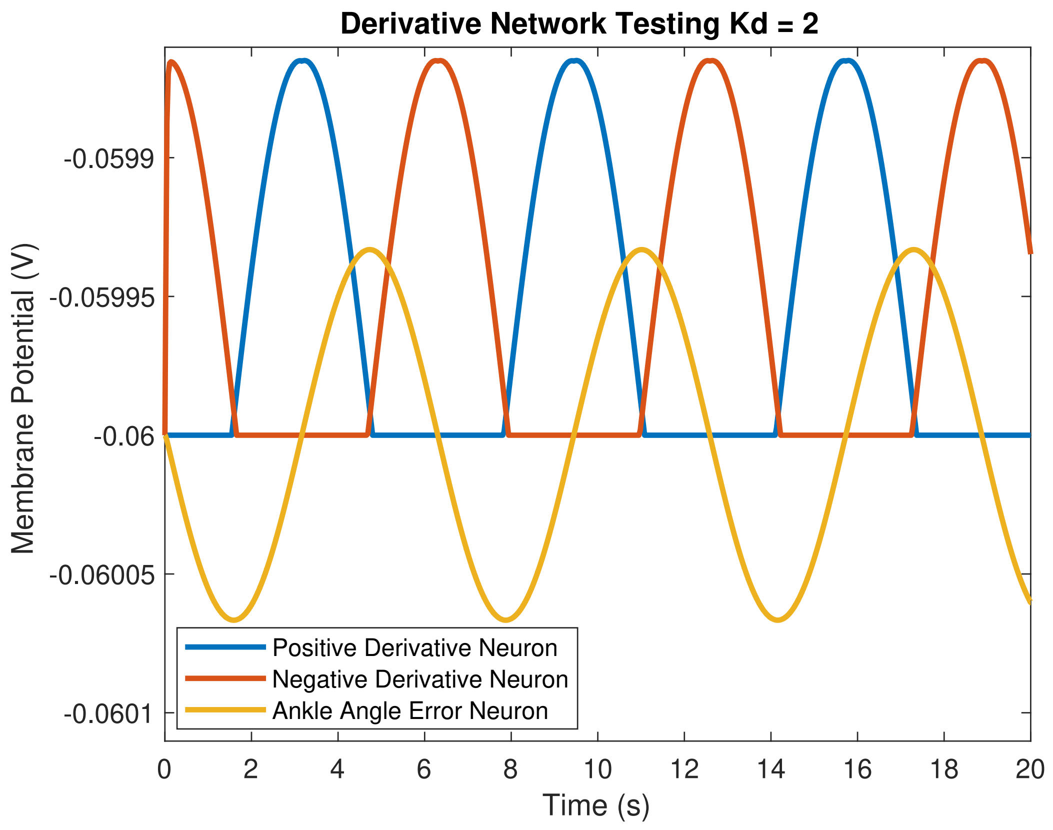


Figure 3.3: Example of a sinusoidal test error signal being processing by a derivative gain subnetwork. The subnetwork was tuned to take the derivative of the error signal and increase the magnitude by a factor of two.

3.1.3 Simulating the Network in Animatlab

The subnetworks outlined above were assembled into a larger network that emulates the classical controller design (Figure 3.1). The network was simulated in Animatlab [4], an open source neuromechanical simulation tool. It provides a powerful environment based in C++ that can perform neural network simulation in real-time. It simulates the same leaky integrator non-spiking neuron model as used in Szczecinski's subnetworks [17]. Animatlab also allows the user to construct a visual model of the network and graphically represent the signal at different points throughout the circuit.

Animatlab has the ability to interface with external devices or software via a serial connection. In order to control the HEBI motor in the inverted pendulum system, the HEBI MATLAB API is used to send torque commands and pull feedback information from the motor.

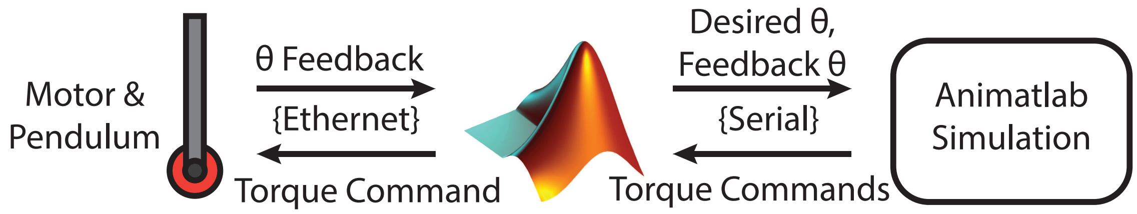


Figure 3.4: Schematic representation of information flow between software and hardware platforms. MATLAB regulates the communication between the motor/pendulum system and the neural controller in Animatlab.

A virtual serial port is used to provide communication between Animatlab and MATLAB. During each iteration of the control loop, MATLAB gets feedback from the motor and writes the current angular position to the serial port going to Animatlab. Animatlab then transforms this position value into an external stimulus current that is injected into Neuron 1. This stimulus current affects the membrane voltage of Neuron 1. The signal is processed by the subnetworks, resulting in the control signal being represented in Neurons 14 and 23's membrane voltages. Animatlab writes the membrane voltage of Neurons 14 (CCW torque) and 23 (CW torque) to the serial port going to MATLAB. This signal is read into MATLAB, transformed, and summed to generate a commanded torque value that is sent to the motor.

Neural Controller Parameter Sensitivity Testing

To develop a better understanding of how sensitive the neural controller is to control parameter variation, I conducted several experiments varying K_p, K_d, K_t and τ_d. The input and output data was processed in the same way as the classical and primary neural controllers' data, using time domain data to derive a transfer function for the controlled system (Section 2.1.5). The parameters were varied to the minimum and/or maximum bounds allowed by the multiplication subnetwork, or until the system became unstable in closed loop control and accurate data could not be collected.

Chapter 4

Results

4.0.1 Primary Results

Time domain data for both the classical and neural closed loop systems was obtained for the white noise experiments. Fitting a 4-poles, 3-zeros transfer function to the this input/output data resulted in the highest degree of accuracy compared to other transfer function forms for both the neural and classical controllers. For the classical controller, the parameters that were found to be the best fit are K_p=11.69, K_d=1.90, K_t=0.0548, ω_c=0.209 and τ=0.0774. The neural network parameters selected to emulate the classical control performance are given in Appendix A.1 and A.2. The neural and classical controllers' gain and phase margin in the frequency domain matched well, having significant overlap and following similar trajectories.

All of the tested and simulated models agreed reasonably well with the phase and gain plots of the human test data, until they diverge near 0.4 Hz. The human data drops in gain and phase much quicker than the other models beyond this threshold. The neural network system response exhibits a slight swell in gain and phase in the higher frequencies between 0.5 and 1.5 Hz that the classical controller simulation and experimental data do not show.

The theoretical model and experimental data for the plant model identification

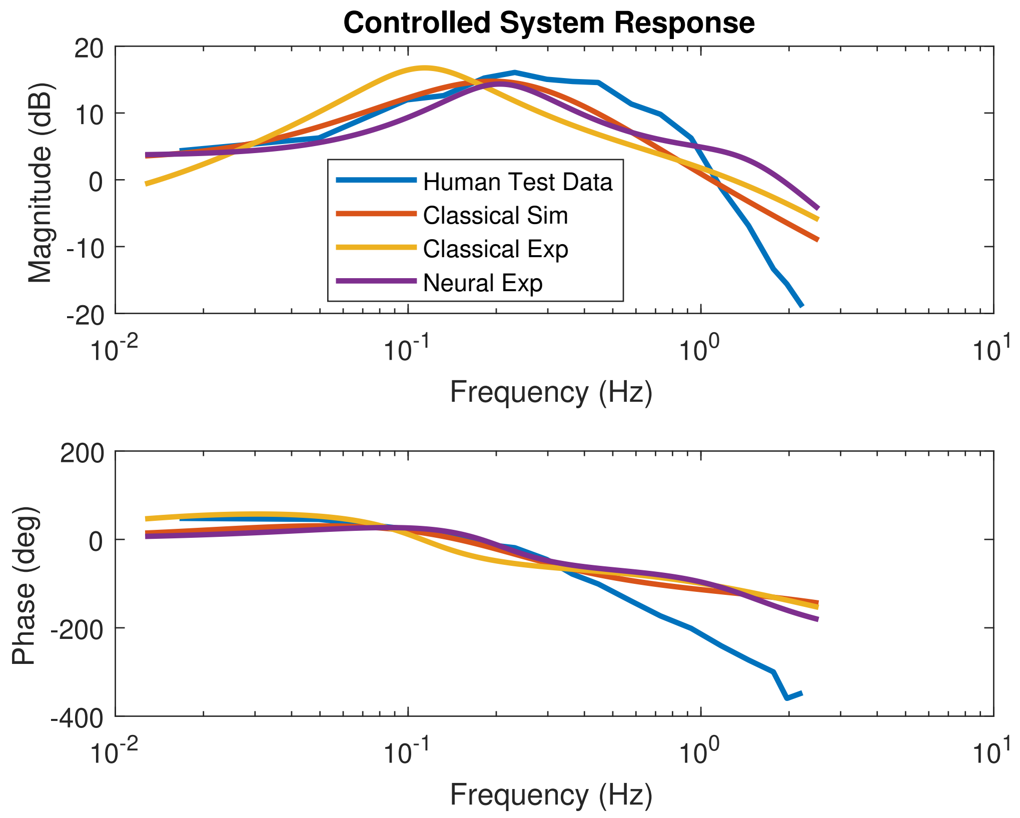


Figure 4.1: Comparison of human test data with the responses of the classical controller design in both simulation and experiment, as well as the response of the neural controller in an experiment

part of this experiment (Section 2.1.2) are compared (Figure 4.2). After running the optimization script, it was found that fixing the theoretical parameters for moment of inertia, J, and destabilizing torque, mgh, and only allowing the damping term, b, to be free placed the model closest to the experimental results. The plant parameters that were used are J=0.44, b=0.40 and mgh=9.50.

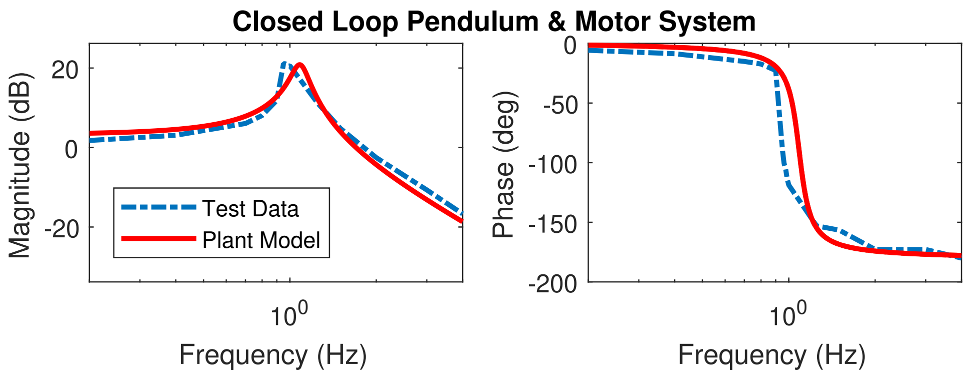


Figure 4.2: Modeling the frequency response of the motor-pendulum system with commanded torque as input and angular position as output. A simple proportional controller was used with a gain of $K_p = 30$. The test data was collected by observing the steady state frequency response of the system at various input frequencies.

4.0.2 Neural Controller Parameter Sensitivity Testing Results

The neural controller has different behavior depending on the gains assigned within the network. I tested how changes in these gains affected the overall system. For clarity of understanding, the multiplication subnetwork gains controlled by a stimulus current within the network have been converted to their classical control parameter equivalents in the proceeding figures.

Decreasing the proportional gain of the controller was found to have a mid-frequency amplitude boosting effect due to the inherently unstable open-loop dynamics of the plant model (Figure 4.3). Higher proportional gain allows the system to adhere to gain values closer to 0 dB at lower frequencies, until a break frequency of the system is reached at just under 2 Hz and the gain drops off. Gains higher than 15.4 caused instability due to the proportional term accelerating the system with insufficient damping to maintain stability. Gains lower than 9.8 were unstable because the proportional contribution of the controller becomes less than the destabilizing torque due to gravity, thus the system cannot return to equilibrium and succumbs to gravitational forces.

The derivative gain values had the greatest effect near the crest of the gain swell from 0.2 to 2.0 Hz (Figure 4.4). Greater damping force from higher K_d values reduced the gain at these frequencies and smoothed out fluctuations in the phase. The derivative term had less gain reduction at lower frequencies where the system velocities are smaller and the positive torque feedback term dominates.

The positive torque feedback gain in the controller was varied from 0 to 1, and the system maintained stability throughout this range (Figure 4.5). While typically positive feedback causes a system to be unstable, the 30 second time constant of the low-pass filter coupled with the positive torque feedback loop allows this feedback to interact with the system without entering instability. The highest K_t value of 1.0 increased the magnitude of the gain peak at approximately 0.17 Hz, but also caused a very steep decrease in gain below 0.1 Hz. Additionally, a significant amount of positive phase is observed at frequencies of 0.1 Hz and lower. When the positive feedback was effectively shut off by setting K_t to zero, the system's resonance peak was completely eliminated, and low frequency gain values were nearly constant at frequencies below 0.5 Hz.

An increase in the time delay parameter resulted in a system with a slightly slower break frequency, and a much larger gain swell approaching the break frequency (Figure 4.6). Longer time delay also increased the phase lag at higher frequencies because the controller reacts slower and falls further behind the commanded position. This effect becomes much more significant at high frequencies where the time delay is larger relative to the input frequency's period. The time delay has little effect on the lower frequencies below 0.1 Hz.

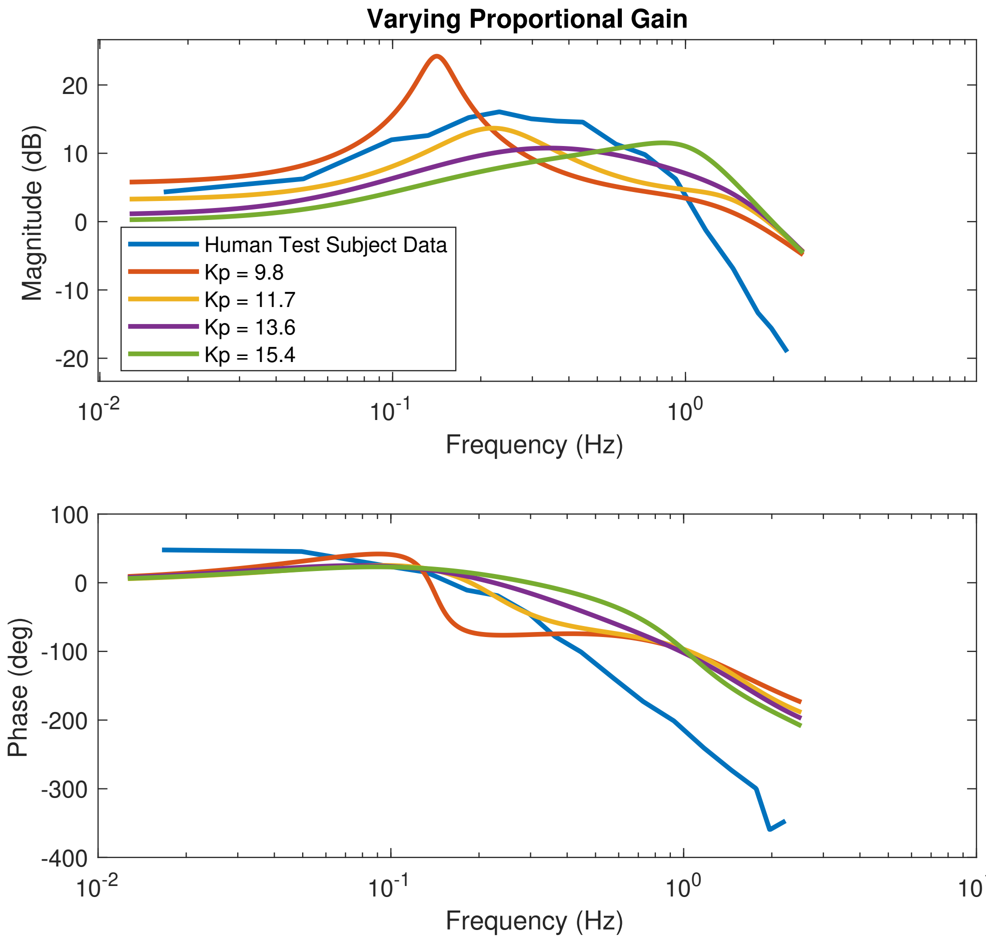


Figure 4.3: The frequency response of the neural controller and motor pendulum system is explored with various proportional gain values (K_p)

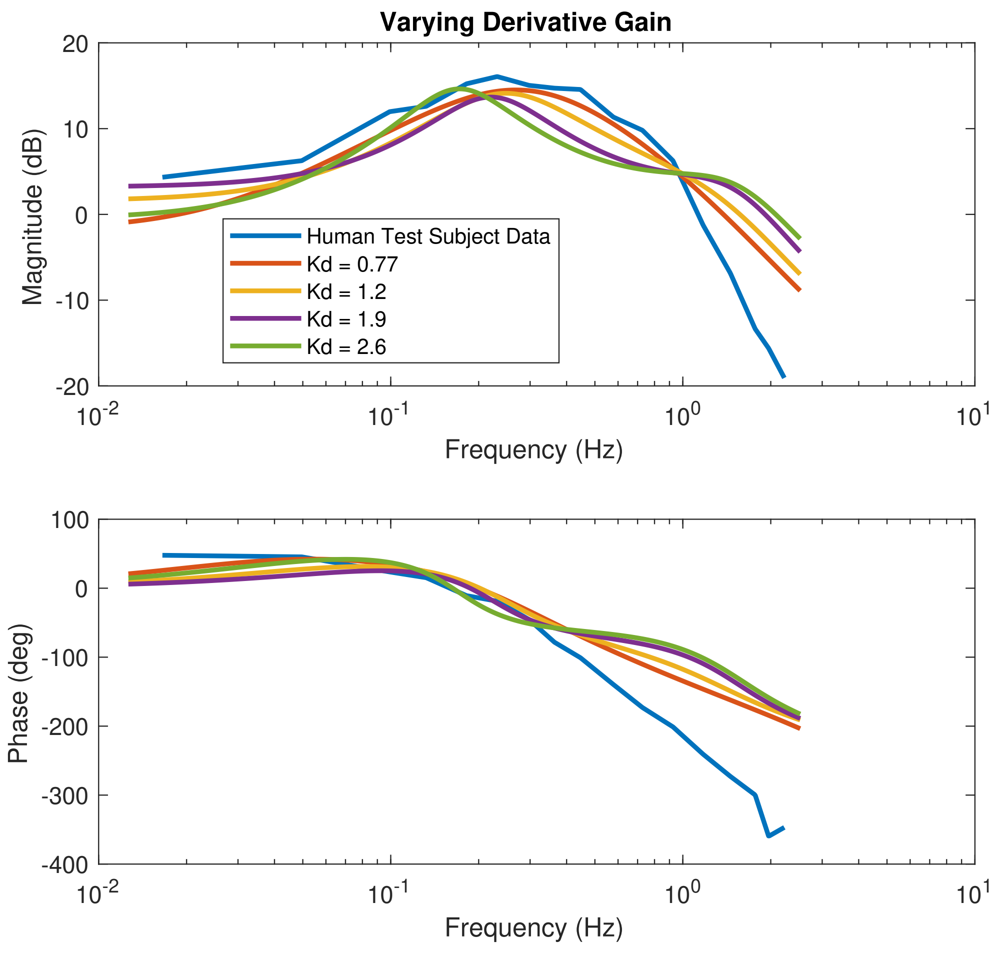


Figure 4.4: The frequency response of the neural controller and motor pendulum system is explored with various derivative gain values (K_d)

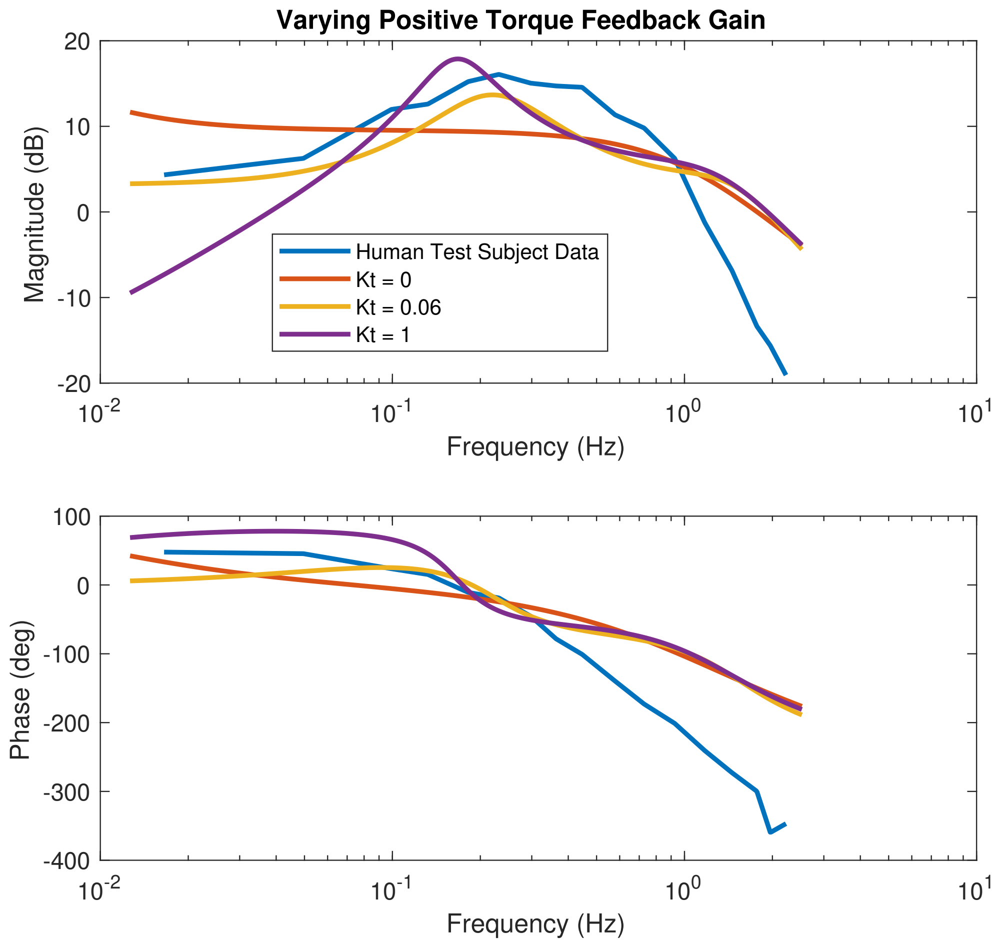


Figure 4.5: The frequency response of the neural controller and motor pendulum system is explored with various positive torque feedback gain values (K_t)

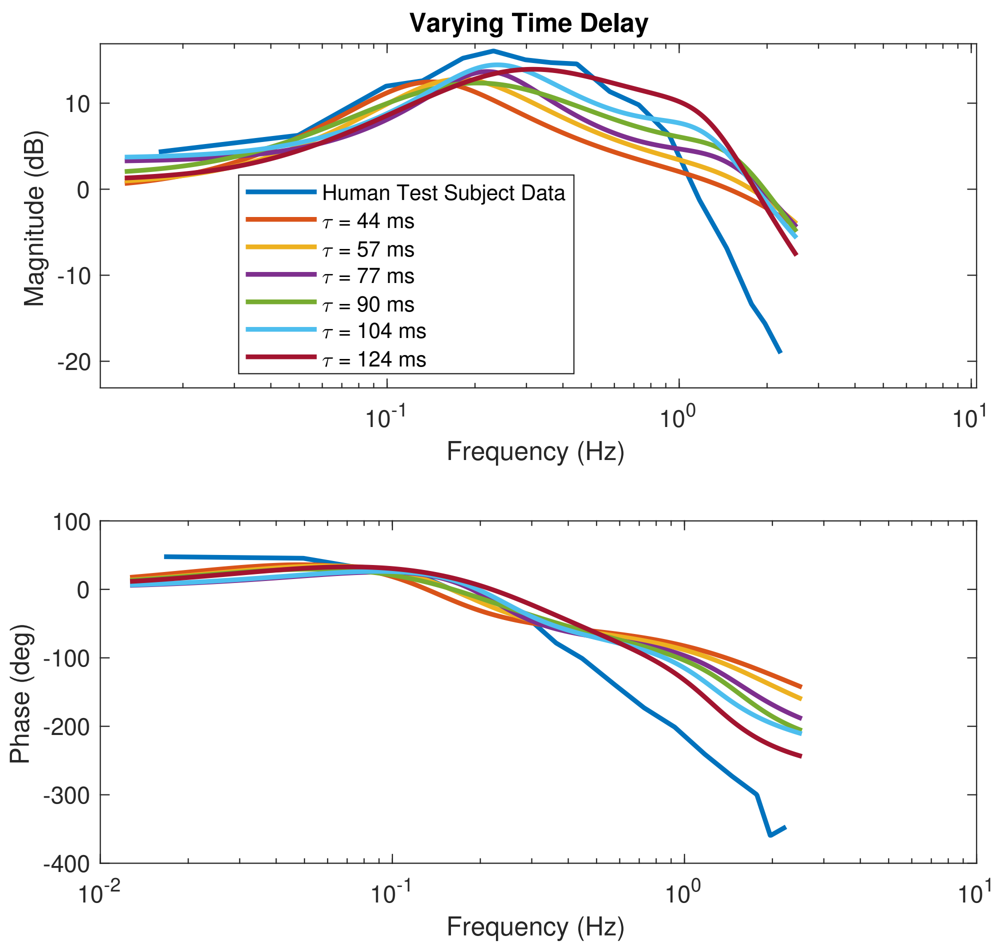


Figure 4.6: The frequency response of the neural controller and motor pendulum system is explored with various time delay values (τ_d)

Chapter 5

Discussion

I hypothesized that the methods presented in this paper would produce a reasonable starting point for developing dynamic neural controllers. I demonstrated this by taking an classical control model fit to test subject data and reproducing this model's behavior using a simple network of neurons. My experiment shows that neural networks are capable of emulating classical control in a specific system, however this process could be generalized to any physical system that can be linearly modeled. It also enables the designer to specify the desired dynamics in these systems and perform system analysis in the well-mapped classical controls domain. Using the methods in this paper, any classical controller can be converted into a neural analog. Combining this neural control architecture with compliant actuators (Festo muscles), biologically inspired joints [15], and physical properties that maximize open loop stability [11], is an exciting path in the field of bio-inspired robotics.

When fitting a transfer function to experimental data, it was found that a 4 poles, 3 zeros transfer function provided a better fit than other transfer function orders. This suggests that my overall theoretical transfer function 2.3 for the system model was an accurate representation. For the controller experiments (Figure 4.1), there are many areas where error can be introduced. Effects such as the time delay, noise, Coulomb friction, and other nonlinearities introduce some error between my test data and model. Any errors in the plant model will propagate into the overall system with a controller. Thus, having an accurate plant model was an important achievement.

The controllers simulated and tested in this paper deviated from human test data more significantly at higher frequencies. This stems from the lack of a feedforward PD component, which increases phase lag and gain drop in this region. The feedforward part of Peterka's model was included to model passive muscle dynamics [1, 12], so it was intentionally omitted in this work as it is an artifact of the physical constraints of the human body - not a deliberate control calculation. I intend to merge physical components and actuators that emulate biological systems into future robotic designs, so capturing these dynamics within my controller is not desirable.

Overall, the plant model provided an excellent fit to the experimental data. This model is crucial to obtaining predictable results when designing a controller. As shown in Figure 4.2, the best fitting second order transfer function has a small amount of deviance from the experimental data. This discrepancy can be explained by the motor's non-linear and/or non-ideal behavior at very low torque values (low frequency input signals) due to Coulomb friction, and also that the $\sin(\theta)=\theta$ linearizing assumption becomes less accurate at high gain values near resonance.

The neural controller and motor/pendulum system was determined to be a fourth order system. With higher order systems, the effects of individual parameters can be complex and interconnected, making it difficult to predict how each parameter might affect the system. In Section 4.0.2, I showed how varying control parameters impacted the frequency response in the neural network. The control parameters were varied by modifying the stimulus currents in the multiplication subnetworks, so all the gains were controlled within the network. This process allows the neural network to process gain adjustments via the stimulus current to these multiplication networks. The continuum of different frequency responses represented in Figures 4.3 through 4.6 shows the flexibility of the neural controller within the biological constraints, and how the controller can produce a range of behaviors under the same general network design.

This neural control design process could be expanded to build a full scale robot with a cascaded control model built off of many neural subnetworks. Centralized vestibular and visual information could be used to determine the desired stabilizing maneuvers that are necessary to achieve the optimal posture for quality sensory feedback and minimize the likelihood of falling over [1]. Engineering models of human balance control that include vestibular and visual feedback loops have already been postulated [12, 13], where the reliance of each of these sensory signals varies depending on their quality. The methods in this paper could be used to design a neural controller that uses multi-sensory feedback. The multiplication subnetwork can set the relative weighting of individual feedback signals and update the gains during operation, based on perceived changes in the quality of the feedback signal. In Figure 3.2, Neurons 8, 12 and 24 act as gain adjustment neurons, where the gain is affected by their membrane voltage. A stimulus current proportional to sensory feedback quality could be applied to similarly functioning neurons that control the gain of a feedback signal (e.g. vestibular feedback) to dynamically adjust the behavior of the controller. In future works, employing a learning algorithm that adjusts gains "on the fly" based on performance metrics of the system and combined feedback from a variety of different sensors is an interesting prospect.

Bibliography

[1] L. Assländer and R. J. Peterka, *Sensory reweighting dynamics following removal and addition of visual and proprioceptive cues*, Journal of Neurophysiology, 116 (2016), pp. 272–285.

[2] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. R. Voelker, and C. Eliasmith, *Nengo: a Python tool for building large-scale functional brain models*, Frontiers in Neuroinformatics, 7 (2014).

[3] M. Chu, B. Kim, S. Park, H. Hwang, M. Jeon, B. H. Lee, and B. G. Lee, *Neuromorphic Hardware System for Visual Pattern Recognition With Memristor Array and CMOS Neuron*, IEEE Transactions on Industrial Electronics, 62 (2015), pp. 2410–2419.

[4] D. Cofer, G. Cymbalyuk, J. Reid, Y. Zhu, W. Heitler, and D. Edwards, *AnimatLab: A 3d graphics environment for neuromechanical simulations*, Journal of neuroscience methods, 187 (2010), pp. 280–8.

[5] F. Corradi, H. You, M. Giulioni, and G. Indiveri, *Decision making and perceptual bistability in spike-based neuromorphic VLSI systems*, in 2015 IEEE International Symposium on Circuits and Systems (ISCAS), May 2015, pp. 2708–2711.

[6] J. A. G. Franco, J. L. d. V. Padilla, and S. O. Cisneros, *Event-based image processing using a neuromorphic vision sensor*, in 2013 IEEE International Autumn Meeting on Power Electronics and Computing (ROPEC), Nov. 2013, pp. 1–6.

[7] A. J. Hunt, M. Schmidt, M. S. Fischer, and R. D. Quinn, *A biologically based neural system coordinates the joints and legs of a tetrapod*, Bioinspiration & Biomimetics, 10 (2015), pp. 055004–055004.

[8] A. J. Hunt, N. S. Szczecinski, E. Andrada, M. Fischer, and R. D. Quinn, *Using Animal Data and Neural Dynamics to Reverse Engineer a Neuromechanical Rat Model*, in Biomimetic and Biohybrid Systems, S. P. Wilson, P. F. M. J. Verschure,

A. Mura, and T. J. Prescott, eds., no. 9222 in Lecture Notes in Computer Science, Springer International Publishing, July 2015, pp. 211–222.

[9] W. Li, N. S. Szczecinski, A. J. Hunt, and R. D. Quinn, *A Neural Network with Central Pattern Generators Entrained by Sensory Feedback Controls Walking of a Bipedal Model*, in Biomimetic and Biohybrid Systems: 5th International Conference, Living Machines 2016, Edinburgh, UK, July 19-22, 2016. Proceedings, N. F. Lepora, A. Mura, M. Mangan, P. F. Verschure, M. Desmulliez, and T. J. Prescott, eds., Springer International Publishing, Cham, 2016, pp. 144–154.

[10] C. Mead, *Neuromorphic electronic systems*, Proceedings of the IEEE, 78 (1990), pp. 1629–1636.

[11] K. Narioka, S. Tsugawa, and K. Hosoda, *3d limit cycle walking of musculoskeletal humanoid robot with flat feet*, in 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, Oct. 2009, pp. 4676–4681.

[12] R. J. Peterka, *Simplifying the complexities of maintaining balance*, IEEE Engineering in Medicine and Biology Magazine, 22 (2003), pp. 63–68.

[13] R. J. Peterka and P. J. Loughlin, *Dynamic Regulation of Sensorimotor Integration in Human Postural Control*, Journal of Neurophysiology, 91 (2004), pp. 410–423.

[14] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, *A Survey of Neuromorphic Computing and Neural Networks in Hardware*, arXiv:1705.06963 [cs], (2017). arXiv: 1705.06963.

[15] A. G. Steele, A. Hunt, and A. C. Etoundi, *Development of a Bio-inspired Knee Joint Mechanism for a Bipedal Robot*, in Biomimetic and Biohybrid Systems, Lecture Notes in Computer Science, Springer, Cham, July 2017, pp. 418–427.

[16] N. S. Szczecinski, A. J. Hunt, and R. Quinn, *Design process and tools for dynamic neuromechanical models and robot controllers*, Biological Cybernetics, 111 (2017), pp. 105–127.

[17] N. S. Szczecinski, A. J. Hunt, and R. D. Quinn, *A Functional Subnetwork Approach to Designing Synthetic Nervous Systems That Control Legged Robot Locomotion*, Frontiers in Neurorobotics, 11 (2017).

Appendix A

Appendix: Neural Network Simulation Parameters and Control Code

A.1 Neuron Parameters Used in Controller

Table A.1: Neuron parameters used in the controller shown in Figure 3.1

Neuron Number	Resting Potential (mV)	Time Const (ms)	Stim Current, I_{app} (nA)
$1,2$	-50	20	0
3,6-7,10,13,15-16,19-21	-60	1	0
$4,17$	-60	0.1	0
$5,18$	-60	8	0
8	-60	1	12.5
$9,12,24$	-60	1	20
11	-60	1	13.5
$14,22$	-60	20	0
23	-60	1	0.8
$25,26$	-60	$30,000$	0

A.2 Synapse Parameters Used in Controller

Table A.2: Synapse parameters used in the controller shown in Figure 3.1

Synapse Number	ΔE (mV)	Conductance, g_s (μS)	Notes
1,3-4,6,9,12-13,19-21, 24-25,29-30,35,37,40-41	194	0.115	Addition
2,7-8,18,23,26,36,39	-40	0.55775	Subtraction Negative
5,22	194	2.2	Mult. K_p
10-11,27-28	194	54	Mult. K_d
14-17,31-34,38	0	20	Mult. Syn 2

A.3 MATLAB Program for Neural Controller

```
%% Wade Hilts - Portland State University
%% Animatlab Serial interface with MATLAB and HEBI MATLAB
    API
clear all;
clc;
% setup serial connection properties
delete(instrfindall)
s = serial('COM3');
set(s,'BaudRate',256000); % it is critical that strict
   baudrate emulation is enabled on the virtual serial
   port driver
s.InputBufferSize=1024;
s.Timeout=15;
fopen(s);
r = serial('COM9');
set(r,'BaudRate',256000); % Animatlab seems to be
   tolerant of very high virtual baudrates. I could have
   gone to 10^6
fopen(r);
% setup HEBI motor API and connection
startup(); % this startup function is called to setup
   HEBI communication
serials = {'X-80103'}; % my HEBI motor serial number
ctrl = CommandStruct();
group = HebiLookup.newGroupFromSerialNumbers(serials);
group.setCommandLifetime(.25);
group.setFeedbackFrequency(150);
HebiLookup
% initialize variables
n=200000; % initializes the maximum number of serial
   bytes that will be read before vectors are full (set
   to well above what sim time needs)
cmd = 1;
cmd_cw = 1;
cmd_ccw = 1;
ct=1;
adv=0;
```

```
CW_T=zeros(n,1); % CW torque commmand vector
CCW_T=zeros(n,1); % CCW torque commend vector
val=zeros(1,n);
var=zeros(1,n);
cmd_sm=zeros(7,1);
cmd_lg=zeros(13,1);
error_adv=zeros(3,1);
chksum=0;
% Control Parameters
gain = 515;
Vout = -.060; %V used to normalize commanded torque
Tout=zeros(n,1);
delay=5; %originally 5
% setup serial package for writing to animatlab
pos_head=[255 255 1 18 0 23 0];%defines header, message
   ID, message size and Data ID
pos_des_head=[24 0];
pos_msg=zeros(1,12);
pos_rad=zeros(1,n);
crit_error = 0;
% Generate filtered Gaussian white noise for sys ID
Fs = 150;
d = fdesign.lowpass('Fp,Fst,Ap,Ast',2,8,.5,60,Fs);
B = design(d,'equiripple');
% create white Gaussian noise the length of your signal
x = 4.*randn(20000,1);
% create the band-limited Gaussian noise
pos_des = filter(B,x)*(pi/180);

% yt=y(1:(length(y)-1));
% Initialize HEBI's logging
% group.startLog();
fbk = group.getNextFeedbackFull();
t0 = fbk.hwTxTime;
% Control Loop below. HEBI motors have a command lifetime
   , so updated
% commands must be sent in less than this time or else
   the motor will turn
% off
while crit_error == 0
```

```
val(ct)=fread(s,1);% read from serial port
switch var(ct)
case 0 % first header value
if val(ct) == 255
var(ct+1) = 1;
end
case 1 % second header value
if val(ct) == 255
var(ct+1) = 2;
end
case 2 % Message ID: data send or error
if val(ct) == 1 % valid message
var(ct+1) = 3;
elseif val(ct) == 2 % error report from animatlab
fprintf('error @ cmd=%d\n',cmd)
error_adv=fread(s,3);
else % some other error has occurred
fprintf('message ID is %d @ cmd %d \n',val(ct),cmd)
end
case 3 % proceed to reading message size
adv=fread(s,1); % advance read position to skip unused
   byte
switch val(ct) % this is the message size
case 12 % small message (1 command)
cmd_sm = fread(s,7);
if cmd_sm(1:2) == [21; 0] % CCW only
CCW_T(cmd_ccw,:) = typecast(uint8(cmd_sm(3:6)),'single');
    % convert 8 bit data to single precision
cmd_ccw = cmd_ccw+1; % advance CCW command index
elseif cmd_sm(1:2) == [22; 0] % CCW only
CW_T(cmd_cw,:) = typecast(uint8(cmd_sm(3:6)),'single');
cmd_cw = cmd_cw+1; % advance CW command index
end
if cmd_sm(7) ~= mod(sum([val((ct-3):ct)'; adv; cmd_sm
   (1:6)]),256) % checksum
disp('checksum error! on cmd_sm')
%                          crit_error =1;
end
case 18 % large message (2 commands)
cmd_lg = fread(s,13);
```

```
CCW_T(cmd_ccw,:) = typecast(uint8(cmd_lg(3:6)),'single');
CW_T(cmd_cw,:) = typecast(uint8(cmd_lg(9:12)),'single');
cmd_cw = cmd_cw+1;
cmd_ccw = cmd_ccw+1;
if cmd_lg(13) ~= mod(sum([val((ct-3):ct)'; adv; cmd_lg
   (1:12)]),256)
disp('checksum error! on cmd_lg')
%                          crit_error = 1;
end
otherwise
fprintf('message size is %d @ cmd = %d \n',val(ct),cmd)
end
% Next, HEBI API is called to obtain position feedback
fbk = group.getNextFeedbackFull();
pos_rad(cmd) = fbk.position;
pos_byte = typecast(single(pos_rad(cmd)),'uint8');
des_byte = typecast(single(pos_des(cmd)),'uint8');
chksum = mod(sum([pos_head pos_byte pos_des_head des_byte
   ]),256);
pos_msg = [pos_head pos_byte pos_des_head des_byte chksum
   ]; % package up serial message
fwrite(r,pos_msg,'uint8'); % write serial message to
   animatlab sim
Tout(cmd)=((CW_T(cmd_cw-1)-Vout)-(CCW_T(cmd_ccw-1)-Vout))
   *-gain; % Commanded Output is defined here
% if statement below controls a time delay between torque
    command realization and command issuing.
% I'm not sure if this works properly as it causes a lot
   of instability.
% I have delay set to zero so commands are instant.
if cmd > delay
ctrl.effort = Tout(cmd-delay);
group.send(ctrl); % this is the HEBI API's function for
   sending motor commands
end
%flush serial buffers
flushinput(s);
flushoutput(r);
cmd=cmd+1; % advance command index
end
```

```
ct=ct+1; % advance serial index
end

%% analyse Animatlab output
% subtraction cicuit output graph:
sub_net=fopen('C:\Users\Wade Hilts\Google Drive\AnimatLab
   \InvertedPendulum_mscfbk_02_26_18_exptest\InputOutput.
   txt','r');
sub_data_headers=textscan(sub_net,'%s\t %s\t %s\t',3);
sub_data=textscan(sub_net,'%f\t %f\t %f\t');
t=sub_data{1};
N_1=sub_data{2};
N_2=sub_data{3};
fclose(sub_net);
%downsample from Ts=.2 to Ts=6.667 ms
N_1_ds = downsample(N_1,33).*(200)+10;
N_2_ds = downsample(N_2,33).*(200)+10;
t_ds = downsample(t,33);
figure;
plot(t_ds,N_1_ds,t_ds,N_2_ds);
legend('Measured','Desired');

%% plot 8 transfer function fits at once
Tsi=0;
load('maple_analysis.mat');


w_plot=logspace(-1.1,1.2,1000);
[mag_n,phase_n,w_n1]=bode(tf1,w_plot);
mag_c1=squeeze(mag_n);
phase_c1=squeeze(phase_n);

[mag_n,phase_n,w_n2]=bode(tf2,w_plot);
mag_c2=squeeze(mag_n);
phase_c2=squeeze(phase_n);

[mag_n,phase_n,w_n3]=bode(tf3,w_plot);
mag_c3=squeeze(mag_n);
phase_c3=squeeze(phase_n);
```

```
[mag_n,phase_n,w_n4]=bode(tf4,w_plot);
mag_c4=squeeze(mag_n);
phase_c4=squeeze(phase_n);

[mag_n,phase_n,w_n5]=bode(tf5,w_plot);
mag_c5=squeeze(mag_n);
phase_c5=squeeze(phase_n);

[mag_n,phase_n,w_n6]=bode(tf6,w_plot);
mag_c6=squeeze(mag_n);
phase_c6=squeeze(phase_n);

[mag_n,phase_n,w_n7]=bode(tf7,w_plot);
mag_c7=squeeze(mag_n);
phase_c7=squeeze(phase_n);

[mag_n,phase_n,w_n8]=bode(tf8,w_plot);
mag_c8=squeeze(mag_n);
phase_c8=squeeze(phase_n);


w_plot=logspace(-1.5,1.5,1000);
[mag_p,phase_p,w_p]=bode(gfr,w_plot);
mag_f=squeeze(mag_p);
phase_f=squeeze(phase_p);

Ts=0;
figure;
subplot(2,1,1)
semilogx(w_p./(2*pi),20*log10(mag_f),w_n1./(2*pi),20*
   log10(mag_c1),...
w_n2./(2*pi),20*log10(mag_c2),w_n3./(2*pi),20*log10(
   mag_c3),...
w_n4./(2*pi),20*log10(mag_c4),w_n5./(2*pi),20*log10(
   mag_c5),...
w_n6./(2*pi),20*log10(mag_c6),w_n7./(2*pi),20*log10(
   mag_c7),...
w_n8./(2*pi),20*log10(mag_c8),'Linewidth',2);
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
```

```
legend('Human Test Subject Data','trial 1','trial 2','
   trial 3','trial 4','trial 5','trial 6','trial 7','
   trial 8','Location','southwest');
title('Controlled System Response');
subplot(2,1,2);
semilogx(w_p./(2*pi),phase_f,w_n1./(2*pi),phase_c1-360,...
w_n2./(2*pi),phase_c2-360,w_n3./(2*pi),phase_c3-360,...
w_n4./(2*pi),phase_c4-360,w_n5./(2*pi),phase_c5-360,...
w_n6./(2*pi),phase_c6-360,w_n7./(2*pi),phase_c7-360,...
w_n8./(2*pi),phase_c8-360,'Linewidth',2);
xlabel('Frequency (Hz)');
ylabel('Phase (deg)');

av_num=(tf1.Numerator+tf2.Numerator+tf3.Numerator+tf4.
   Numerator+tf5.Numerator+tf6.Numerator+tf7.Numerator+
   tf8.Numerator)./8;
av_den=(tf1.Denominator+tf2.Denominator+tf3.Denominator+
   tf4.Denominator+tf5.Denominator+tf6.Denominator+tf7.
   Denominator+tf8.Denominator)./8;

av_exp_Kt_20=idtf(av_num,av_den);
```

A.4 Greybox State Space Model for Classical Controller in MATLAB

```
%% Wade Hilts - idgrey call-out function
function [A,B,C,D] = Human_Cntrl(Kp,Kd,Kt,wc,tau,a,b,c,Ts
   )
% Human balance control model
c4=(tau*a+Kt*wc*tau*Kd*a);
c3=(Kt*wc*tau*Kp*a+Kt*wc*tau*Kd*b+2*a-tau*Kd-2*Kt*wc*Kd*a
   +b*tau+tau*wc*a);
c2=(Kt*wc*tau*Kd*c+b*tau*wc+tau*c+Kt*wc*tau*Kp*b+2*b+2*wc
   *a-2*Kt*wc*Kd*b-tau*Kp-2*Kt*wc*Kp*a-tau*Kd*wc+2*Kd);
c1=(2*c-tau*Kp*wc-2*Kt*wc*Kd*c+Kt*wc*tau*Kp*c+tau*wc*c+2*
   Kd*wc-2*Kt*wc*Kp*b+2*Kp+2*wc*b);
c0=2*c*wc+2*Kp*wc-2*Kt*wc*Kp*c;

b3=-tau*Kd;
b2=(2*Kd-tau*Kp-tau*Kd*wc);
```

```
b1=(2*Kp-(-2*Kd+tau*Kp)*wc);
b0=2*Kp*wc;
% set up matrices
% X0=aux(4:end);
A=[0, 1, 0, 0; 0, 0, 1, 0; 0, 0, 0, 1; ...
-c0/c4, -c1/c4, -c2/c4, -c3/c4];
B=[0 0 0 1/c4]';
C=[b0, b1, b2, b3];
D=0;

% K=zeros(8,1);

if Ts>0
s = expm([[A B]*Ts; zeros(2,10)]);
A = s(1:8,1:8);
B = s(1:8,9:10);
end

end
```

A.5 Greybox State Space Plant Model in MATLAB

```
%% Wade Hilts - idgrey call-out function
function [A,B,C,D] = OLTF_graybox(a,b,c,Kp,Ts)
% define parameters common to all subjects
A=[0, 1; -(c+Kp)/a, -b/a ];
B=[0 Kp/a]';
C=[1 0];
D=0;
end
```

A.6 Plant System ID MATLAB Code

```
%% Wade Hilts - Portland State University - Fall 2017
% Motor-Pendulum System Identification
% pendulum is T shape, with T orthogonal to axis of
   rotation (larger J)
clc;
```

```
clear all;
% close all;
startup();
HebiLookup
serials = {'X-80103'};
group = HebiLookup.newGroupFromSerialNumbers(serials);
cmd = CommandStruct();
group.setFeedbackFrequency(150);


n=1000000;
cmd_pos=zeros(n,2);
act_pos=zeros(n,2);
tor_cpl=zeros(n,2);
ts=1;
error=0;
error_m1=0;
tor_c=0;
%
Kpi=30;
Kd=0;
count=0;

w = 2*pi*.975;
amp = 3*(pi/180); % 8 degrees pp

group.startLog();
fbk = group.getNextFeedbackFull();
t0 = fbk.hwTxTime;
while true
fbk = group.getNextFeedbackFull();
t = fbk.hwTxTime - t0;
count=count+1;
pos_c = amp*sin(w*t);
error = pos_c - fbk.position;
%     cmd_pos(ts,1:2)=[t pos_c];
%     act_pos(ts,1:2)=[t fbk.position];
tor_c = Kpi*error + Kd*error_m1;
%     tor_cpl(ts,1:2) = [t tor_c];
cmd.effort = tor_c;
```

```
group.send(cmd);
ts=ts+1;
error_m1=error;
if t > 140*(2*pi/w)
break
end
end
log=group.stopLogFull();
figure;
plot(log.time(2:end),diff(log.time));
figure;
plot(log.hwTxTime(2:end),diff(log.hwTxTime));
figure;
plot(log.time, log.effort);
% figure;
hold on
cmd_pos(ts,1:2) = [t pos_c];
plot(log.time, amp.*sin(w.*log.time))
plot(log.time, log.position);
legend('torque','cmd position','act position')
%% Compile FR data
% close all;
s=tf('s');
% removed first decade of static system behavior in freq
   dom
% f=[.01 .03 .08 .1 .2 .4 .7 .8 .9 .95 .9625 .975 .9875 1
    1.25 1.5 2 3 4 6];
f=[ .2 .4 .7 .8 .9 .95 .9625 .975 .9875 1 1.25 1.5 2 3
   4];
% dt=[226.6-225.1 108.8-108.3 78.31-78.08 62.63-62.47
   51.31-51.23 45.68-45.62...
%     47.55-47.49 49.11-49.05 49.23-49.16 42.63-42.37
   40.03-39.74 39.53-39.23 ...
%     39.06-38.74 41.58-41.25 30.14-29.8 33.13-32.84
   31.36-31.12 27.24-27.08 ...
%     24.435-24.31 20.46-20.38];
dt=[51.31-51.23 45.68-45.62...
47.55-47.49 49.11-49.05 49.23-49.16 42.63-42.37
   40.03-39.74 39.53-39.23 ...
39.06-38.74 41.58-41.25 30.14-29.8 33.13-32.84
```

```
   31.36-31.12 27.24-27.08 ...
24.435-24.31];

cmd_pp=ones(length(f),1)'*2*amp;
% act_pp=[.05789+.06845 .06217+.06444 .06105+.06527
   .06131+.065 .06247+.06595 ...
%      .07647+.07278 .1049+.1052 .1343+.1305 .2075+.2036
   .6031+.5838 .6047+.5803 ...
%      .5885+.5699 .5627+.5436 .5645+.5509 .1921+.1858
   .09472+.08714 .04088+.03731 ...
%      .01639+.01427 .01008+.005246 .005429+.001605];
act_pp=[ .06247+.06595 ...
.07647+.07278 .1049+.1052 .1343+.1305 .2075+.2036
   .6031+.5838 .6047+.5803 ...
.5885+.5699 .5627+.5436 .5645+.5509 .1921+.1858
   .09472+.08714 .04088+.03731 ...
.01639+.01427 .01008+.005246];

dB=20.*log10(act_pp./cmd_pp);
w=(2*pi).*f;
ph=dt.*f*-360;

Kpi=30;

Ai=.44;
Bi=.4;% bi=.9;
Ci=-9.5;

OLTF=1/(Ai*s^2+Bi*s+Ci);
CLTF_f=feedback(Kpi*OLTF,1);

Ts=0;
s=tf('s');
sysID=120.9/(s^2+3.349*s+103.8);
w_plot=logspace(.1,1.4,1000);
[mag_sys,phase_sys,w_sys]=bode(CLTF_f,w_plot);
mag_s=squeeze(mag_sys);
phase_s=squeeze(phase_sys);

figure;
```

```
subplot(1,2,1)
semilogx(w./(2*pi),dB,'Linewidth',2,'Linestyle','-.');
hold on;
semilogx(w_sys./(2*pi),20*log10(mag_s),'Linewidth',2,'
   Color','r')
hold off;
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
legend('Test Data','Plant Model','Location','southwest');
title( '

   Closed Loop Pendulum & Motor System');
subplot(1,2,2);
semilogx(w./(2*pi),ph,w_sys./(2*pi),phase_s,'Linewidth'
   ,2,'Linestyle','-.');
hold on;
semilogx(w_sys./(2*pi),phase_s,'Linewidth',2,'Color','r')
   ;
xlabel('Frequency (Hz)');
ylabel('Phase (deg)');

data=(act_pp./cmd_pp).*exp(1j*ph*pi/180);
gfr = idfrd(data,w,Ts);
% fit=procest(gfr,'P1D');
load('BVL_SurfaceTilt_1deg_EyesClosed.mat');
clear tf
s=tf('s');

Ktot=5;
wn=23;
z=1.2;

Kpi=30;
%
% Ai=.433;
% Bi=.6;% bi=.9;
% Ci=-10.48;


% OLTF=1/(I*s^2+frc*s-m*g*h);
```

```
OLTF=1/(Ai*s^2+Bi*s+Ci);
% CLTF_w=(Ktot*wn^2)/(s^3+2*z*wn*s^2+(wn^2)*s+Ktot*wn^2);


CLTF_f=feedback(Kpi*OLTF,1);
CLTF=feedback(OLTF,1);
CLTF_d=c2d(CLTF_f,.002,'tustin')

%%
% mass of pendulum: ~3.36 kg
% moment of inertia: ~0.433 kg*m^2
% COM height: .318 m
Kpi=30;
% Ti=0.0005;
Ai=.433;% bi=.9;
Bi=.4;
Ci=-10.25;

Tsi=0;
par={'a',Ai;'b',Bi;'c',Ci;'Kp',Kpi};
aux={};
% use idgrey function to generate data comparison

TF_fit = idgrey('OLTF_graybox',par,'cd',aux,Tsi);
Ap=TF_fit.a;
Bp=TF_fit.b;
Cp=TF_fit.c;
% generate estimated Kp, Kd, Kt, wc, tau

TF_fit.Structure.Parameters(1).Free = true;
TF_fit.Structure.Parameters(2).Free = true;
TF_fit.Structure.Parameters(3).Free = true;
TF_fit.Structure.Parameters(4).Free = true;
%     TF_fit.Structure.Parameters(5).Free = false;
%     human_fit.Structure.Parameters(9).Free = false;
TF_fit.Structure.Parameters(1).Maximum = 1.1*Ai;
TF_fit.Structure.Parameters(1).Minimum = .9*Ai;
TF_fit.Structure.Parameters(2).Maximum = 2.5*Bi;
TF_fit.Structure.Parameters(2).Minimum = .2*Bi;
TF_fit.Structure.Parameters(3).Maximum = .8*Ci;
TF_fit.Structure.Parameters(3).Minimum = 1.2*Ci;
TF_fit.Structure.Parameters(4).Maximum = 1.05*Kpi;
TF_fit.Structure.Parameters(4).Minimum = .95*Kpi;


TF_est = greyest(gfr,TF_fit)

% compare(gfr,human_fit,hum_cntrl_est)
% Plot results

w_plot=logspace(-1,2,1000);
[mag_p,phase_p,w_p]=bode(gfr,w_plot);
mag_f=squeeze(mag_p);
phase_f=squeeze(phase_p);

[mag_n,phase_n,w_n]=bode(TF_fit,w_plot);
mag_c=squeeze(mag_n);
phase_c=squeeze(phase_n);

[mag_n1,phase_n1,w_n1]=bode(TF_est,w_plot);
mag_c1=squeeze(mag_n1);
phase_c1=squeeze(phase_n1);

figure;
subplot(2,1,1)
semilogx(w./(2*pi),dB,w_n./(2*pi),20*log10(mag_c),w_n1
   ./(2*pi),20*log10(mag_c1),'Linewidth',2);
% semilogx(w./(2*pi),dB,w_n./(2*pi),20*log10(mag_c),'
   Linewidth',2);

xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
legend('Test Data','Unfit Model','Model Fit','Location','
   southwest');
% legend('Test Data','Model','Location','southwest');
title('Closed Loop');
subplot(2,1,2);
semilogx(w./(2*pi),ph,w_n./(2*pi),phase_c,w_n1./(2*pi),
   phase_c1,'Linewidth',2);
% semilogx(w./(2*pi),ph,w_n./(2*pi),phase_c,'Linewidth
    ',2);
xlabel('Frequency (Hz)');
ylabel('Phase (deg)');
% figure;
% pzmap(TF_est)
```