Brigham Young University
BYU ScholarsArchive

All Theses and Dissertations

2016-08-01

Comparing Efficacy of Different Dynamic Models for Control of Underdamped, Antagonistic, Pneumatically Actuated Soft Robots

Morgan Thomas Gillespie
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Mechanical Engineering Commons

BYU ScholarsArchive Citation

Gillespie, Morgan Thomas, "Comparing Efficacy of Different Dynamic Models for Control of Underdamped, Antagonistic, Pneumatically Actuated Soft Robots" (2016). *All Theses and Dissertations*. 5996.
https://scholarsarchive.byu.edu/etd/5996

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.

Comparing Efficacy of Different Dynamic Models for Control of Underdamped, Antagonistic, Pneumatically Actuated Soft Robots

Morgan Thomas Gillespie

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Marc D. Killpack, Chair
Timothy W. McLain
Steven K. Charles

Department of Mechanical Engineering

Brigham Young University

August 2016

Copyright © 2016 Morgan Thomas Gillespie

All Rights Reserved

ABSTRACT

Comparing Efficacy of Different Dynamic Models for Control of Underdamped, Antagonistic, Pneumatically Actuated Soft Robots

Morgan Thomas Gillespie
Department of Mechanical Engineering, BYU
Master of Science

Research in soft robot hardware has led to the development of platforms that allow for safer performance when working in uncertain or dynamic environments. The potential of these platforms is limited by the lack of proper dynamic models to describe or controllers to operate them. A common difficulty associated with these soft robots is a representation for torque, the common electromechanical relation seen in motors does not apply. In this thesis, several different torque models are presented and used to construct linear state-space models.

The control limitations on soft robots are induced by natural compliance inherent to the hardware. This inherent compliance results in soft robots that are commonly underdamped and present significant oscillations when accelerated quickly. These oscillations can be mitigated through model-based controllers which can anticipate these oscillations. In this thesis, multiple model predictive controllers are implemented with the torque models produced and results are presented for an inflatable single-DoF pneumatically actuated soft robot.

Larger, multi-DoF, soft robots present additional issues with control, where flexibility in one joint impacts control in others. In this thesis a preliminary method and results for controlling multiple joints on an inflatable multi-DoF pneumatically actuated soft robot are presented.

While model predictive controllers are capable, their control commands are defined by solving an optimization constrained by model dynamics. This optimization relies on minimizing the cost of a user-defined objective function. This objective function contains a series of weights, which allow the user to tune the importance of each component in the objective function. As there are no calculations that can be performed to tune model predictive controllers to achieve superior control performance, they often need to be tuned tediously by a skilled operator. In this thesis, a method for automated discrete performance identification and model predictive controller weight tuning is presented.

This thesis constructs multiple state-space models for single- and multi-DoF underdamped, antagonistic, pneumatically actuated soft robots and shows that these models can be used with model predictive control, tuned for performance, to achieve accurate joint position control.

Keywords: inflatable robot, robotics, dynamic model, MPC, model predictive control, thesis

ACKNOWLEDGMENTS

I would like to acknowledge and give thanks for the love and support provided by my family throughout my time as a student. My wife, Courtney, provided the encouragement and sympathy needed to get through the long hours and difficult evenings required to complete this work. She has helped me proofread all of my papers and even served as technical support when problems inevitably arose outside of my expertise.

I would like to extend a special thanks to Dr. Marc Killpack, my graduate adviser, a fantastically dedicated lab head who took a great risk in bringing me into his lab in 2014 as a student without any research experience. He has been extremely helpful in all facets of my life as a student and engineer. His exhaustive work ethic and enjoyable personality make him a delight to work for and with. He strives very hard to ensure we are never left wanting the proper equipment to complete our work. Without him and his guidance this area of work would not be anywhere close to where it is today.

I would also like to thank my lab mates and friends who have helped me throughout the years. They have made the Robotics and Dynamics Laboratory an enjoyable place to be and have improved the overall quality of my work. We have worked together on many projects and publications that were only achievable through close collaboration.

I gratefully acknowledge support from the NASA Early Career Faculty grant NNX14A051G, which has made this research possible. I also recognize the hard work of Kevin Albert and his team at Pneubotics in building and supporting the soft robot hardware platform used within this work.

Lastly I would like to thank Dr. David Wingate who provided assistance in understanding neural network topology as well as how to apply it specifically to our system. He also graciously provided both software and hardware assistance when it was sorely needed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

Developments in soft pneumatic structures and actuation systems have produced physically capable soft robots, able to lift a payload comparable to a similarly sized rigid robot, but at a fraction of the overall system weight. One type of these newer robots utilizes pressurized internal structural bladders and antagonistic actuation bladders contained within high-strength fabric. While capable, these pressurized bladders have high inherent compliance through their compressible air structure and actuation. This compliance in links, joints, and structure, as well as the unique actuation method, requires new dynamic models capable of describing the torque generated at each joint and also the system flexibility. These light and compliant robots are inherently safer to operate around people but will require significant research to improve their usability to that of more rigid platforms.

1.1 Contributions

This thesis documents the author's contributions to this body of research in creating multiple dynamic models and controllers for soft pneumatic robotic platforms. The following is a list of major achievements described within this thesis:

- Development of an impedance-based linear state-space dynamic model for an antagonistic pneumatically actuated robot which considers pressure dynamics.
- Development of a deep neural network capable of predicting future states of a pneumatic system.
- Utilization of a deep neural network to construct a linear state-space model.
- Development of a model predictive controller capable of commanding both joint position and target pressures which correlate to stiffness.

- Extension of the model predictive controller to use a joint-torque model, impedance model, and neural network based model.
- Extension of the model predictive controller and models to multiple degrees of freedom showing improved performance compared to previous work.
- Development of an automated tuning tool for model predictive controllers using standard performance characteristics.

1.2 Motivation and Related Works

The unique advantages of soft, inflatable robots over rigid robots for specific applications were what motivated this research. Applications included health care, living assistance, space exploration, search and rescue, orthotics, and prosthetics. It had already been shown that contact forces from inflatable links could be controlled in [1] and [2] with cable-driven actuators. In [3] the Head Injury Criterion (HIC) was identified as a limit for serious head injury upon robot impact, where inertia of the robot is a major driving contributor. This indicates that with all operating parameters constant that a reduction in robot inertia directly reduces the HIC rating. This reduction in inertia was possible with an inflatable fabric-based soft robot joint.

Designs for rotary, fabric-based, pneumatically actuated joints were proposed in [4] and the actuators used in this work were based on these designs. Work in [5] characterized different models for traditional rigid servo-pneumatic actuators which made different constant temperature assumptions. In [6] these assumptions were used to control force and stiffness for a linear pneumatic actuator. A different actuation approach utilizing McKibben Artificial Muscles, which emulate human muscles, was proposed in [7] and [8]. More recently, work has been completed on rotary elastic chamber actuators such as in [9] and [10], where two antagonistic bellows imparted torque on an armature rotating about a rigid rotary joint. In all of these cases the safety and robustness introduced by compliant joints was mitigated by higher-inertia motors or rigid links between the joints. While these models were both useful and accurate for these rigid systems, they did not carry over well to systems with significant flexibility or non-rigid structure.

The greatest difficulty with modeling these platforms was attributed to the lack of a relation between pressure and torque. This thesis describes the torque output of inflatable antagonistic joints through multiple methods, including impedance modeling, torque characterization, and modern advancements and applications of neural networks. Through this modeling of joint torque, this thesis sought to construct relatively generic dynamic models and optimal control methods that could be used to control any flexible rotary joint with a non-rigid structure.

There is a lack of literature on the modeling and control of inflatable structures. However, the wide range of potential applications suggest a viable area for new research.

CHAPTER 2. BACKGROUND

This chapter seeks to provide an understanding of the background information required for comprehension of the content within this thesis. A description of the specific robotic platform used for this research is provided to better understand why current dynamic models are not very applicable. A description of model predictive control is provided as it is the primary control methodology used throughout. Lastly a description of deep neural networks is provided as they are one of the modeling methods utilized within this work.

2.1 Platform Description

While the models derived within this work are intended to be general, platform-specific considerations were made throughout and a detailed description of the platform provided additional clarification. The inflatable robot platforms used in this research were designed and constructed by Pneubotics, an affiliate company of Otherlab.

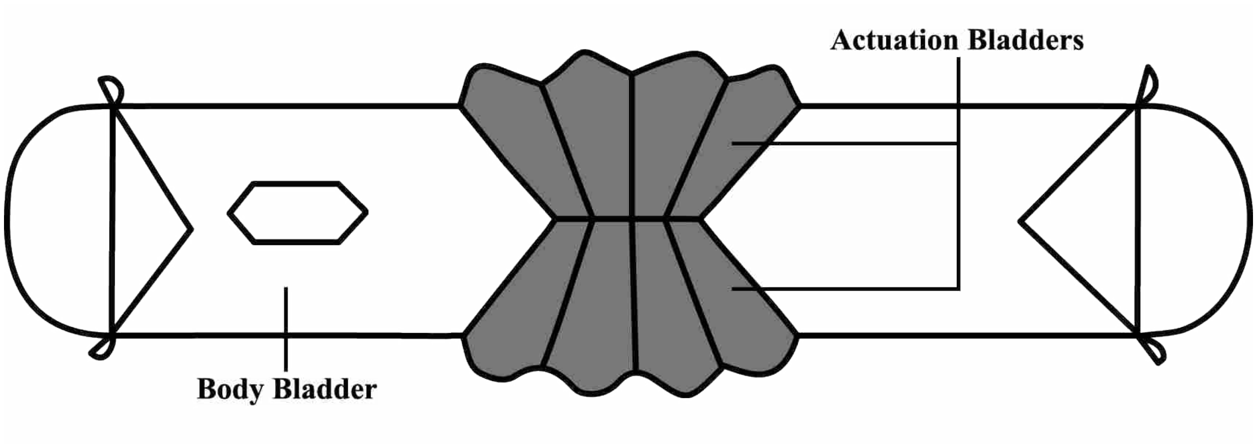

Figure 2.1: Pneumatic joint construction.

Inelastic ballistic nylon fabric was sewn together to form the outer shell of a robot and internal polyurethane bladders were inflated to press against the skin, giving the robot structure without any internal skeleton. Joints were designated by two opposing, flexible, inextensible pockets sewn into the outer shell, seen as grey in Figure 2.1, where the actuation bladders were inserted. Bidirectional bending was achieved by pressurizing the actuation bladders, causing them to expand and lengthen. The bladder pocket fabric contained this expansion and translated extension into joint rotation by imparting force on the internal structural bladder. The internal body bladder was inflated to 1-2 pounds per square inch gauge (psig) pressure whereas the actuation bladders were filled to 0-20 psig. The source pressure was provided by a 100 psig house compressor fed into a local 30 gallon air storage tank at 80 psig with a regulated output of 25 psig.

Air flow was controlled from the pressure source and vented to atmosphere through Enfield LS-v25 5-port spool valves. For the purpose of the work, only one output port of the Enfield valves was used, which made these valves act as 3-port spool valves. As seen in Figure 2.2, each bladder had an individual valve for control, while sharing the same pressure source.

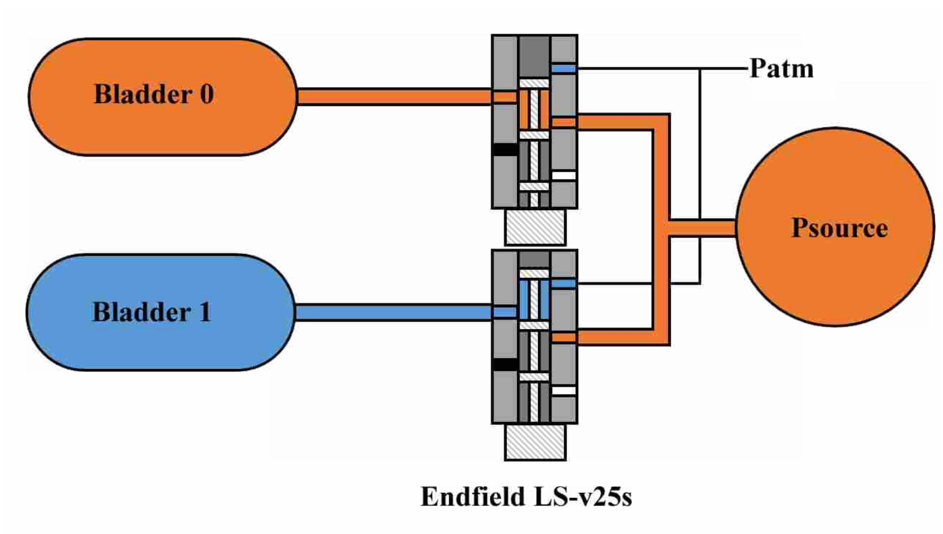

Figure 2.2: Representative figure of valve and bladder configuration.

Pressure within the actuation bladders was read by Honeywell HSCDRNN100PGSA3 0-100 psig pressure transducers. These pressure sensors shared a circuit board with an InvenSense MPU-9150, a 9-axis Inertial Measurement Unit (IMU) containing a 3-axis accelerometer, gyro, and magnetometer. These sensor boards were secured within protective enclosures and mounted to the inelastic robot skin through velcro straps.

Communication between the controller and sensors was achieved using the Robot Operating System (ROS) version Indigo operating in non-real time on an Ubuntu workstation. Sensor values for the IMU and pressure were read and published at approximately 1 kHz. Pressure for each bladder was controlled by an underlying proportional-integrator-derivative (PID) controller operating at approximately 1 kHz directly controlling valve current. Desired pressures were published over ROS and the valves were actuated to achieve commanded pressure.

2.1.1 Single Degree of Freedom: Grub

The single Degree of Freedom (DoF) test bed platform, named the Grub as seen in Figure 2.3, was used for the majority of model and controller development. One end of the exterior structure was secured to a wooden platform along with the valves. A single, unactuated body bladder extended throughout the entire length, pressing against the non-rigid outer shell to provide structural strength. The internal body bladder was bent at its midsection during actuation, bisecting the central bladder into two adjoined links at the same pressure.

Robot pose measurement was achieved through an IMU board affixed to the outside of the top body link. Joint angle was determined by the IMU orientation relative to the world gravity vector, which was assumed to be normal to the base platform.

2.1.2 Multiple Degrees of Freedom: King Louie

King Louie, seen in Figure 2.4, had two four-foot-long arms each with five flexible joints, a torso with two joints that allowed forward-back and side-side movements, and two grippers at the end of the arms resulting in 14 degrees of freedom. King Louie's arms utilized the same actuator design present in the Grub. These joints were sewn together in series, often perpendicular to the previous joint. This stacking of five inflatable rotary joints resulted in a soft, flexible, and lightweight robotic platform capable of lifting loads comparable to that of similarly-sized flexible robots, such as the Baxter platform from Rethink Robotics.

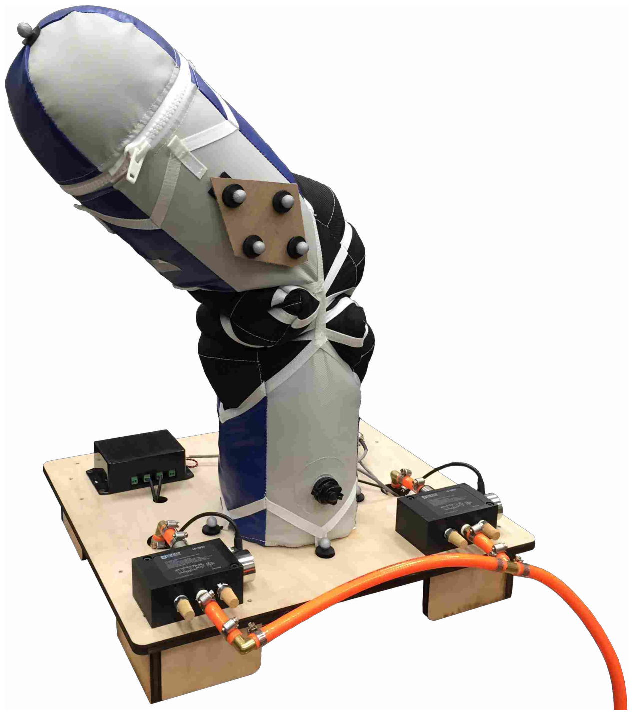
Figure 2.3: Grub single joint robot

Due to the nature of the joints, direct joint angle measurement through traditional rotary encoders was difficult. As such, joint angles were instead estimated through link pose measurements. For the purpose of these calculations, the links were assumed to be rigid, as most of the system compliance was in the joints. With an IMU mounted on each link, it should be possible to estimate all joint angles through relative orientations. Work within this thesis instead measured link orientation with a motion capture camera system, which calculated relative orientation between links and estimated joint angles online. The motion capture system used eight Kestrel digital cameras operating at up to 300 Hz. The motion capture data was processed by MotionAnalysis Cortex software and published over a local network.

Joint angles were calculated using the kinematic relationship between rigid links, described with Denavit-Hartenberg (DH) parameters in Table 2.1.

Using these DH parameters, the orientation of a distal link could be calculated by using kinematic relations between links. The orientation of the distal link is calculated by forward multiplied rotational transformation matrices describing the kinematic relations by the pose of the proximal link.

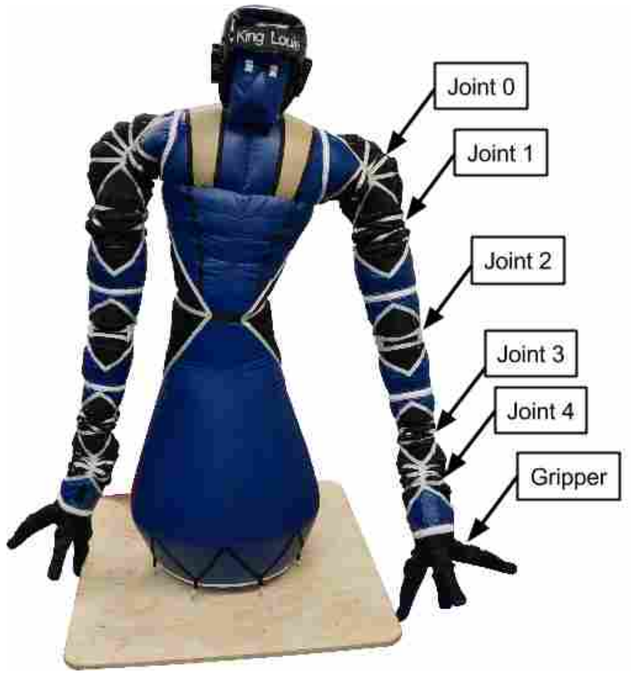

Figure 2.4: King Louie multi-DoF robot

Table 2.1: DH parameters for King Louie.

	Base-Joint 0	Joint 0-Joint 1	Joint 1- Joint 2	Joint 2-Joint 3	Joint 3- Joint 4	Joint 4-Gripper
DH_θ	$\frac{\pi}{2}$	$\theta_1+\frac{\pi}{2}$	θ_2	θ_3	θ_4	$\theta_5+\pi$
DH_d	0	-0.05	0	0	0	0
DH_a	0	0.18	0.32	0.28	0.14	0.28
DH_α	$\frac{\pi}{2}$	$-\frac{\pi}{2}$	0	0	$\frac{\pi}{2}$	0

An example calculation is provided below, where two rigid links, denoted by their orientations $P[n]$ and $P[n+1]$, were connected through two stacked rotary joints: $R[n]$ which laid adjacent to link n and $R[n+1]$ which laid adjacent to link $n+1$. Joints $R[n]$ and $R[n+1]$ were connected by an infinitely short link connected between them that was ignored. Using pose information for link $P[n]$ and kinematic relations, such as DH parameters, the pose of $P[n+1]$ could then be calculated with the joint angles describing $R[n+1]$ and $R[n]$ as seen in Equation 2.1.

$$P[n+1] = R[n+1]R[n]P[n] \tag{2.1}$$

where $P[n+1]$ is the orientation of a distal link $n+1$ expressed as a 3x3 rotational matrix relative to ground, $R[n+1]$ is the 3x3 rotational matrix describing the rotation of the joint closest to link $n+1$, $R[n]$ is the 3x3 rotational matrix describing the rotation of the joint closest to link n, and $P[n]$ is the orientation of link n expressed as a 3x3 rotational matrix relative to ground.

This process was general and could be used with any two orientations connected by a known kinematic chain, including orientations measured by two IMUs. The motion capture system was used to only track links with IMUs mounted within them, this was done so that this process of joint angle estimation could be expanded to include the IMU pose estimation in future work.

Each arm contained an individual body bladder extending through all five joints. A separate, much larger, central body bladder served to support the torso and ran between the waist actuators. For the purpose of this thesis, the waist actuators were unactuated and instead filled to stabilize the torso.

2.2 Model Predictive Control

Model predictive control (MPC) is a finite horizon optimal control strategy that relies on mathematical models to predict future states across a finite horizon to calculate an optimal control trajectory across this horizon.

MPC originated from the process industry in chemical production and oil refinement [11] and has been used more recently in robotics [12–16] and UAV research [17–19]. Modern advances in computing power and optimization techniques such as those proposed in [20,21] made high-rate MPC solving possible and opened its usage up to areas in robotic controls.

MPC allows for the minimization of configurable costs across a discrete finite time horizon given specific constraints. This minimization is achieved through an optimization solver constrained by the system dynamics. MPC not only allows for model-based control but also the consideration of real-world constraints such as joint limits, pneumatic valve actuator restrictions, and hardware pressure limits.

At each time step, the MPC optimization solver takes in the current states and solves a convex cost function subject to the model dynamics and other specified constraints. For the purpose of this work, discrete linear dynamics were used within the solver to describe the plant. Model dynamics in this work were considered linear and time-invariant as they were kept constant over the predictive horizon, despite being updated at every time step. Using plant dynamics, MPC predicts the states over a specific time horizon by varying the inputs, solving for an array of inputs that minimizes the specified cost function. Once a solution is found, the first time step of the solution is applied to the system and newly measured states are fed into the solver and the problem is solved again at the next time step.

An efficient solver for the MPC problem described in this work in Section 4.1 was generated using CVXGEN [22], a web-based tool for developing convex optimization solvers. The optimization solver was written in C and Python code was used to called the solver.

2.3 Deep Neural Networks

Deep neural networks (DNN) are a series of mathematical function models inspired by the human brain to define any arbitrary non-linear function. In the human brain, a seemingly random network of neurons receives a series of inputs, often sensory information, and activates neurons in a sequence to produce the desired output, such as muscle activation based on sensed pressure in the skin.

DNNs can be used to approximate complex functions with unknown dynamics, known as universal function approximation. These nets are described as a series of interconnected nodes which exchange information upon activation. These connections are usually numerical values weights tuned to match the desired system that are summed and fed into transfer functions. Nodes are traditionally assembled in layers, as seen in Figure 2.5. The number of nodes and layers required is an open problem and often the size of the net is selected through trial and error to achieve best results.

Any number of inputs can be fed into a specifically constructed net and run through a series of internal layers and calculations, resulting in a predetermined number of outputs. Internal nodes that are neither input nor output nodes are referred to as hidden, seen as nodes 1, 2, and 3 in Figure 2.5.

As a sample calculation, the output of Node 1 in Figure 2.5 would be calculated as:

$$H_1 = Input_1 * w11 + Input_2 * w21 + Bias * w01 \tag{2.2}$$

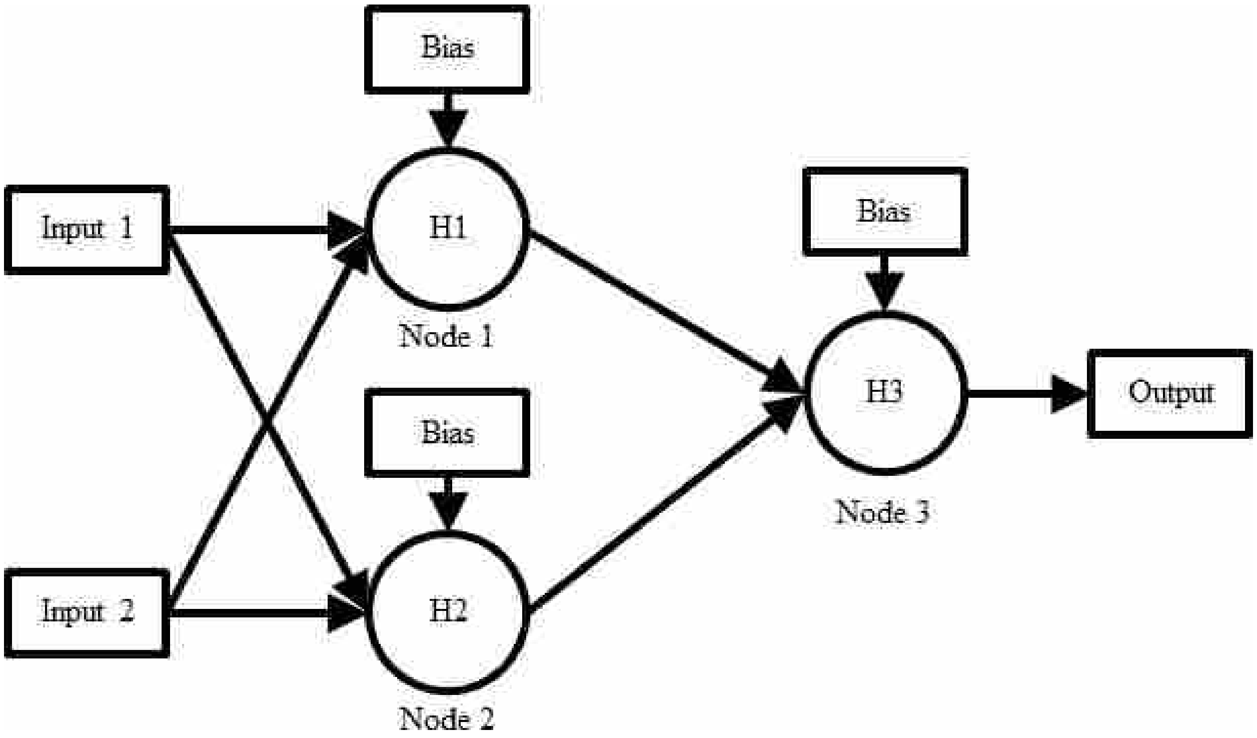

Figure 2.5: Example neural network structure

$$Output_1 = \frac{1}{1+e^{-H_1}} \tag{2.3}$$

where H_1 is the internal value of the node, $w11$ is the weight from input 1 to node 1, $w21$ is the weight from input 2 to node 1, $w01$ is the weight of the bias, and $Output_1$ is the output of the node H1 and input to node H3. The bias is a constant value, often equaling 1, which serves to give the network additional flexibility.

When initially constructed, the values of these weights are assigned randomly, producing vastly incorrect outputs. To adapt the net and produce the desired outputs, a process known as training, large amounts of data are needed. Usually more data points than total nodes is required for training, where a singular data point is a known input matched with a known output. For the purpose of training in this research, a large collection of known inputs to outputs were generated and randomly divided into two groups: training and validation data, at a ratio of 10 to 1, respectively. The training data inputs were fed into the network and then the results were compared to the known outputs. Error was calculated as the difference in the resultant output to the correct output. The error was then used along with a process known as backpropagation, an abbreviation of backwards propagation of errors, to tune the weights. The weights from each node in the hidden layer connected to the outputs were changed as a function of the error and a set learning rate. The learning rate determined how rapidly the weights should be changed. This process of changing the weights to reduce error was continued throughout the entire structure, one layer at a time, until the input layer was reached.

The errors for each output of the training data set were combined into a single value, often mean squared error (MSE), and compared against the MSE of the separate validation set. This validation set was also run through the net and used to check for overfitting. This process of backpropagation was then repeated until the training and validation MSE diverged significantly, seen in Figure 2.6 as the red dashed line.

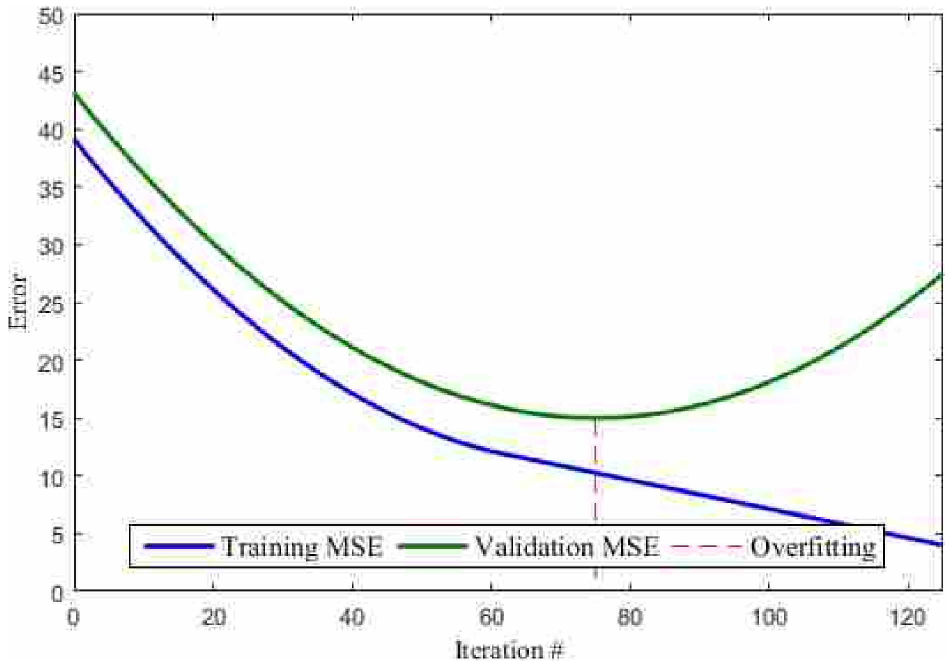

Figure 2.6: Example plot of error demonstrating overfitting.

This point of divergence of validation and training MSE serves as an acceptable stopping point for the optimization, where any additional matching to the training data would make the DNN less general.

The machine learning library Google Tensorflow version 0.9.0 was used within this work for the production of all neural networks. Advancements in computational speed, graphic processor assisted computation, and ease of use opened this software up for more general uses, include high-rate robotic modeling.

CHAPTER 3. MODEL DEVELOPMENT

In this chapter various models describing torque generated in inflatable actuators, such as those mentioned in Chapter 2, are derived and formulated. These models seek to improve state prediction for usage within model predictive control (MPC). Since MPC relies heavily on the model dynamics to predict future states, a more accurate model should provide better predictions and control performance.

The work in Section 3.1 was completed by Best et al. in [23] but an understanding of the developed model is required for the context of the other models described. The author of this thesis was the primary researcher for work in [24] described in section 3.2. The work of Section 3.3 was collaborative and the author of this thesis contributed through model development, control implementation, and the publication in [25]. The deep neural network (DNN) model described in Section 3.4 was produced by the author of this thesis with assistance from the Perception, Control, and Cognition Laboratory at Brigham Young University, led by Dr. David Wingate.

3.1 2-State Impedance

In some of the earliest work on these inflatable systems [26], the links were modeled as an inverted rigid pendulum with angular velocity, $\dot{\theta}$, and angle, θ, as states:

$$I\ddot{\theta} + K_d\dot{\theta} + K_s\theta - mg\frac{L}{2}sin(\theta) = \tau \tag{3.1}$$

where I is the calculated moment of inertia rotating about the approximate joint center, K_d is a damping constant, m describes the mass of the link, g describes gravity, and L is the approximate distance from the joint center to the center of mass.

Through dynamic parameter identification, the value for m was found to be near zero and 3 orders of magnitude below similarly optimized values for K_d and K_s. Due to the lightweight inflatable hardware, the influence of mass on the system performance was identified to be negligible and was removed from the single-DoF model.

To begin the modeling process, a steady-state joint angle response for a range of different pressure inputs was produced. A range of inputs were applied to the system and the steady-state angle was recorded; between each point the system was drained completely. These input-to-state relations were plotted as a surface plot. This plot, seen in Figure 3.1, produced a general trend to relate steady-state angle and pressures in both bladders.

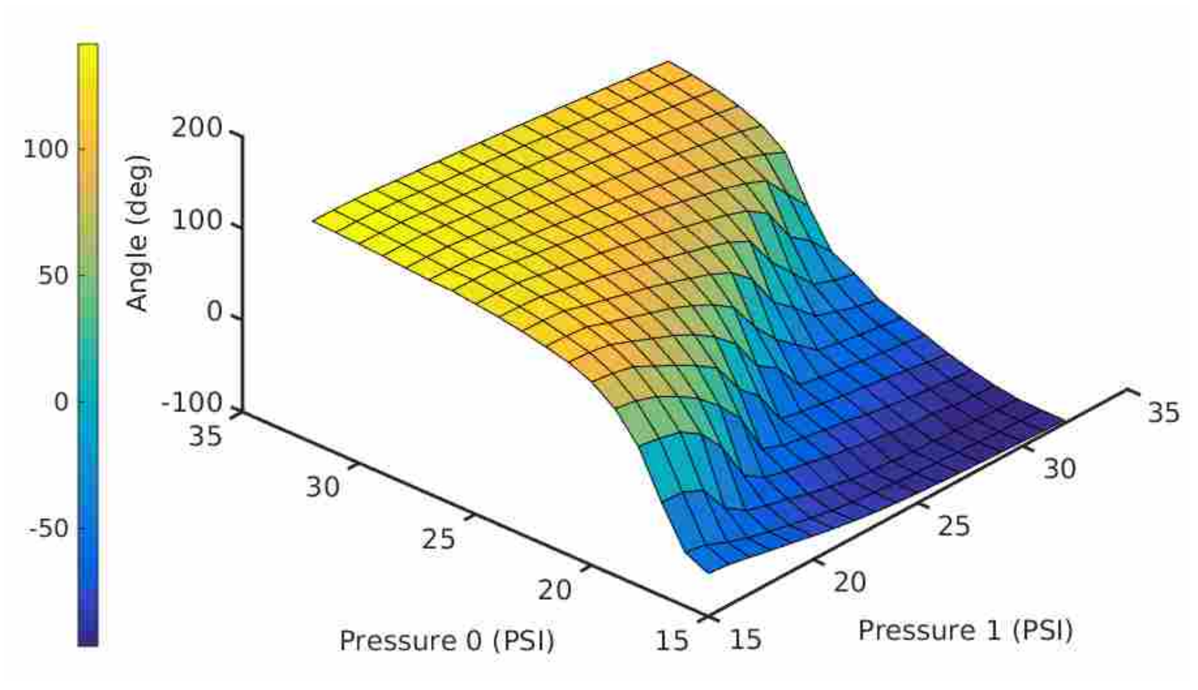

Figure 3.1: Plot of steady state mapping used to convert equilibrium angles to chamber pressures.

In an effort to reduce the number of degrees of freedom for control, the 3D map was reduced to a 2D function by taking an effective cross sectional slice between the minimum and maximum θ values and projecting the entire surface onto this cross sectional plane. This curve, seen in Figure 3.2, related the angle θ to a single variable P, which runs along the cross sectional plane. The values below and above the fit sigmoid highlight the severe hysteresis present in this system.

This simplification reduced the multi input - P_0 and P_1, single output - θ (MISO) system to a single input - P, single output - θ (SISO), which made the model and controller simpler to derive and implement. A sigmoid function was fit to the 2D cross section seen in Figure 3.2.

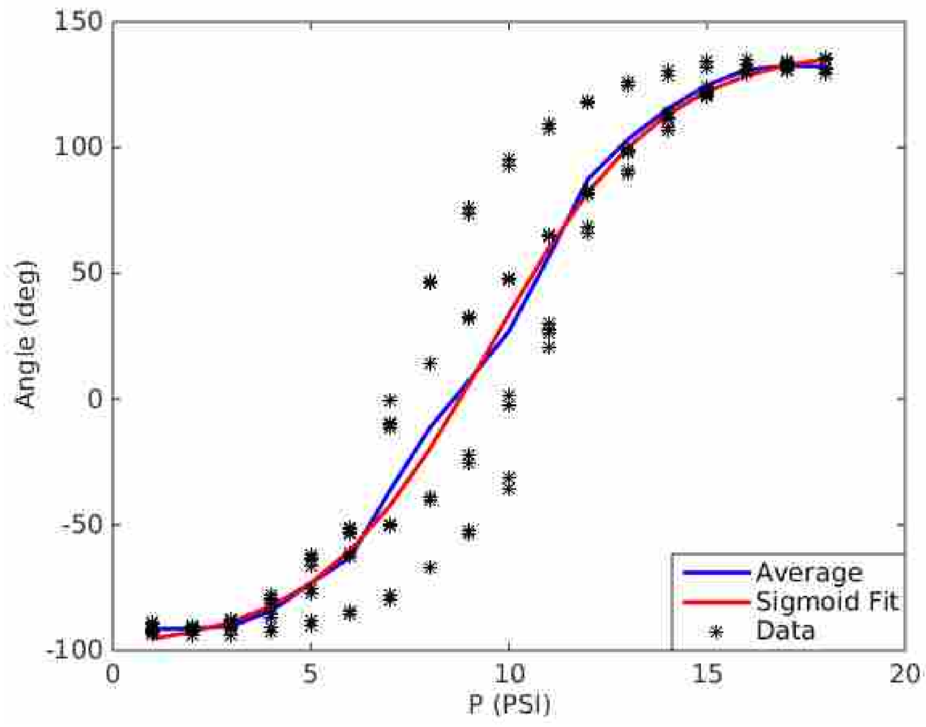

Figure 3.2: 2D cross section plot of steady state angle to single variable P.

Since no known torque to pressure relation existed for these platforms and the platform appeared to match the dynamics of a spring damper system, an impedance model was used to describe torque, with the sigmoid function fit between θ and P to calculate an equilibrium angle. This equilibrium angle was the angle the system would settle to based upon given input pressures, provided the chambers were drained and then filled to the input pressures.

This equilibrium angle was used along with the current angle state, θ, in an impedance model to describe torque at the joint as a torsional spring:

$$\tau = K_s(\theta_e - \theta) \tag{3.2}$$

where K_s is a fixed stiffness constant, θ_e, is the equilibrium angle determined by the pressure to angle map, seen in Figure 3.2.

These equations were then be placed in linear state-space form:

$$\begin{bmatrix} \ddot{\theta} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} -\frac{K_d}{I} & -\frac{K_s}{I} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \theta \end{bmatrix} + \begin{bmatrix} \frac{K_s}{I} \end{bmatrix} \begin{bmatrix} \theta_e \end{bmatrix} \tag{3.3}$$

3.2 4-State Planar Impedance

While the model described in Section 3.1 was sufficient for angle tracking, the work only utilized change in angle and angular velocity as states and did not consider the dynamics of the underlying pressure controller dynamics for each actuation chamber.

The dynamics of the underlying PID pressure controller were captured by a simple first order model:

$$\dot{P} = -aP + bP_D \tag{3.4}$$

where P is the pressure in a corresponding bladder, P_D is the desired pressure, and a and b are parameters fit to data collected for commanded steps in pressure.

This model was verified by holding one actuator at constant pressure while step inputs in P_D were commanded to the opposing bladder through the underlying pressure PID controller. The model developed in Equation 3.4 was simulated in open-loop using the measured data and commanded inputs. The resultant simulation was plotted against measured data in Figure 3.3, which shows an acceptable match for the PID dynamics. For the purpose of this work, fill and drain rates were assumed to be equal and higher order dynamics were ignored. Perfectly modeling the true difference in fill and drain rates would require better gas dynamic models, compensating for choked flow, and a model of the valve aperture, which is not instrumented.

The link dynamics for the model described next were constructed similarly to the 2-state model in Section 3.1, starting with the simplified dynamic model of an inverted pendulum described again ignoring gravity:

$$I\ddot{\theta} + K_d\dot{\theta} = \tau \tag{3.5}$$

with an impedance torque representation:

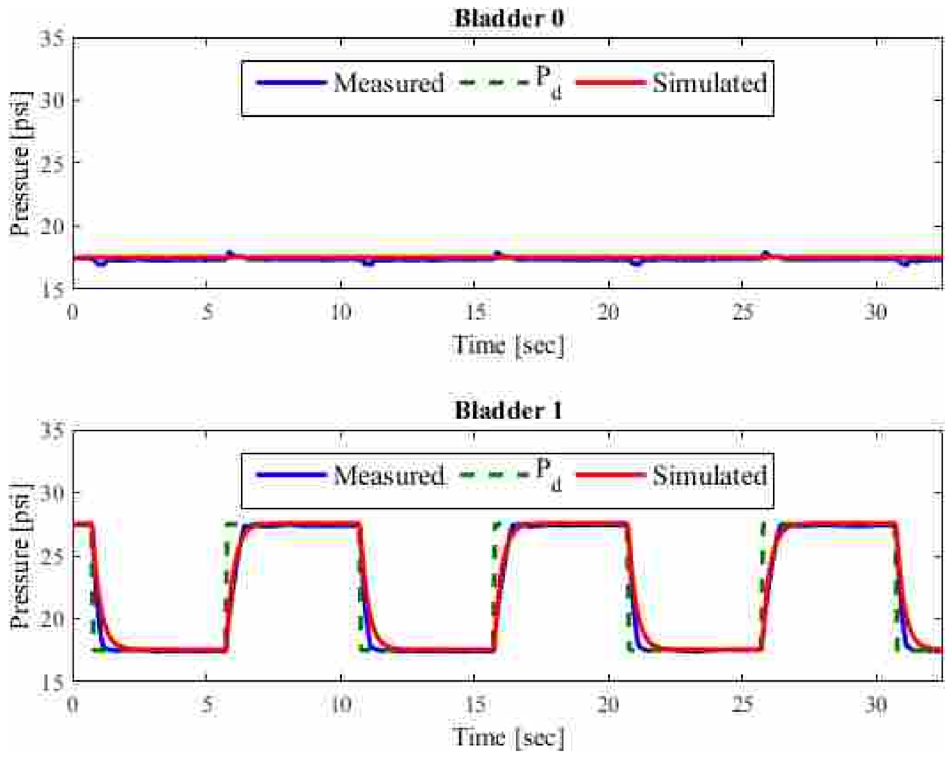

Figure 3.3: Simulated first-order pressure dynamics

$$\tau = K_s(\theta_e - \theta) + K_d\dot{\theta} \tag{3.6}$$

This impedance representation assumes that the damping term K_d within the torque representation can be combined with the K_d within the system dynamics.

Instead of using the nonlinear experimental equilibrium mapping between P_0, P_1, and θ shown in Figure 3.1, a tangent planar estimation taken from a smoothed version of that surface was used at each time step, seen in Figure 3.4. The original surface was smoothed and refined offline using the Matlab interp2 function, configured with cubic spline interpolation, which increased grid density by a factor of four.

This planar equation was constructed as a function of the pressures in the two antagonistic bladders, P_0 and P_1:

$$\theta = \alpha_1 + P_0\alpha_2 + P_1\alpha_3 \tag{3.7}$$

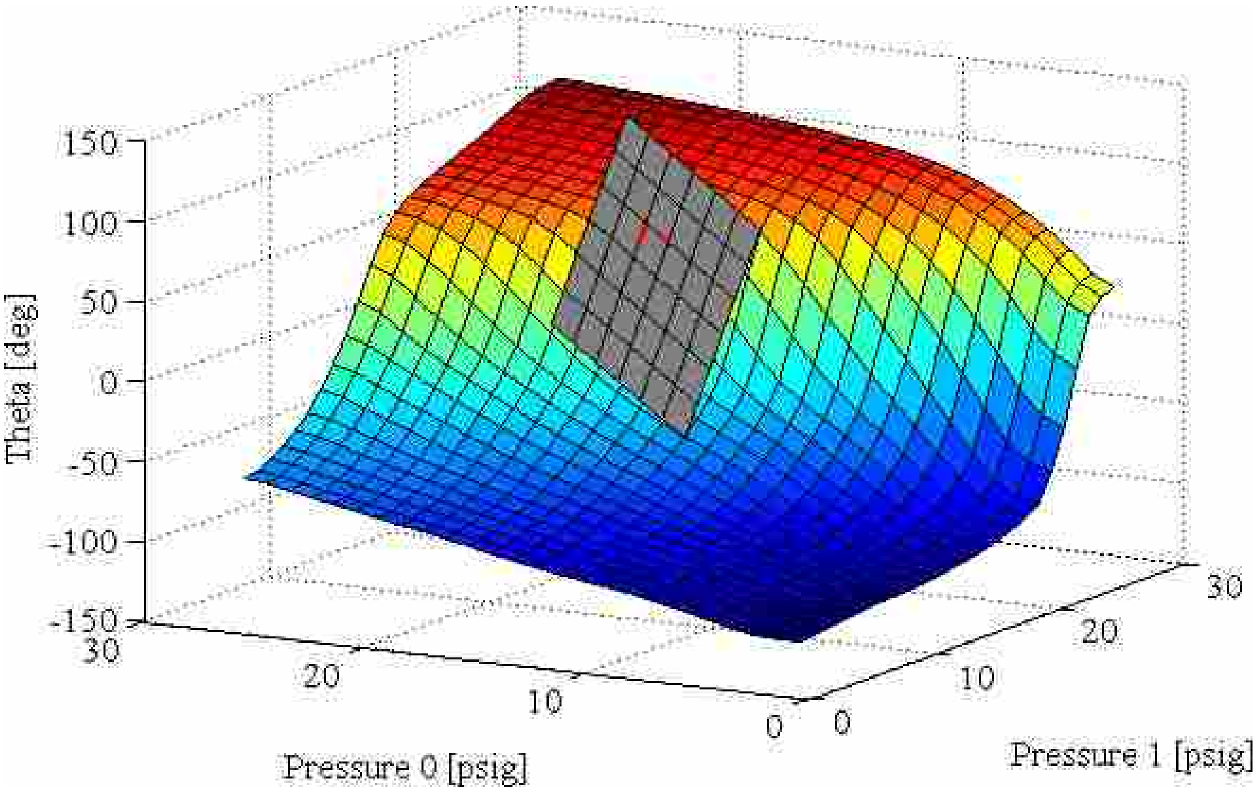

Figure 3.4: Tangent plane plotted atop θ surface map.

where α_1, α_2, and α_3 are three parametric constants needed to describe a 3D plane, P_0 is the pressure for one bladder, and P_1 is the pressure for the opposing bladder.

While α_1, α_2, and α_3 are considered functions of P_0 and P_1, they are notated as constants in the system dynamics as they are kept constant over the predictive horizon used in control and updated at every time step.

The variable θ_e was also approximated using this same planar relation, using the previously acquired α_2 and α_3 along with commanded pressures sent to the underlying pressure PID controller as inputs:

$$\theta_e = \alpha_1 + P_{D,0}\alpha_2 + P_{D,1}\alpha_3 \tag{3.8}$$

where $P_{D,0}$ and $P_{D,1}$, represent the desired pressures for bladder 0 and 1 respectively.

Commanded pressures were used to describe θ_e as they are the pressure values that the system will reach at steady state.

By combining Equations 3.6, 3.5, 3.7, and 3.8, a differential equation, which describes the link dynamics as a function of current and commanded pressures, can be constructed.

$$I\ddot{\theta} + K_d\dot{\theta} = K_s(\alpha_2(P_{D,0} - P_0) + \alpha_3(P_{D,1} - P_1)) \tag{3.9}$$

In Equation 3.7 α_1 serves to describe the vertical offset of the planar surface approximated by the steady-state angle map. By considering α_1 in both θ and θ_e, this offset is removed from the system dynamics. The removal of α_1 serves to dismiss the influence of measurement error and hysteresis when comparing the steady-state angle to current system configuration or state.

The rigid-body dynamics in Equation 3.9 along with pressure dynamics in Equation 3.4 can then be placed in linearized state-space form:

$$\begin{bmatrix} \ddot{\theta} \\ \dot{\theta} \\ \dot{P}_0 \\ \dot{P}_1 \end{bmatrix} = A \begin{bmatrix} \dot{\theta} \\ \theta \\ P_0 \\ P_1 \end{bmatrix} + B \begin{bmatrix} P_{D,0} \\ P_{D,1} \end{bmatrix} \tag{3.10}$$

where:

$$A = \begin{bmatrix} \frac{-K_d}{I} & 0 & \frac{-K_s\alpha_2}{I} & \frac{-K_s\alpha_3}{I} \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -a & 0 \\ 0 & 0 & 0 & -a \end{bmatrix} \tag{3.11}$$

$$B = \begin{bmatrix} \frac{-K_s\alpha_2}{I} & \frac{-K_s\alpha_3}{I} \\ 0 & 0 \\ b & 0 \\ 0 & b \end{bmatrix} \tag{3.12}$$

In this formulation, the matrix A is singular. As such, the matrix exponential method that was used in [26] could not be used for discretization. However, the controllability matrix formed from this system was full rank for A and B in the form of Equations 3.11 and 3.12 indicating the system was still controllable. These state-space equations were instead transformed from the continuous time-domain to discrete state space equations using the bilinear transform method.

This transformation provided the discrete state space equations:

$$\begin{bmatrix} \dot{\theta}[k+1] \\ \theta[k+1] \\ P_0[k+1] \\ P_1[k+1] \end{bmatrix} = A_d \begin{bmatrix} \dot{\theta}[k] \\ \theta[k] \\ P_0[k] \\ P_1[k] \end{bmatrix} + B_d \begin{bmatrix} P_{D,0}[k] \\ P_{D,1}[k] \end{bmatrix} \tag{3.13}$$

These discrete-time state-space equations could then be used for predicting the future states of the system given pressure inputs. Equation 3.13 served as constraint within the model predictive controller described in Chapter 4. This discretization method was used for all subsequent models described in this chapter.

3.3 Torque Model

Further modeling efforts were made in [25] to develop a model utilizing experimental torque measurements. This model was developed under the idea that each bladder produced an independent torque on a joint which was a function of the pressure in the bladder and the current angle of the arm. The resultant difference between these torques and the inherent joint stiffness produced the total joint torque τ_a that was seen by the links, causing acceleration.

To experimentally identify these torque contributions, the base link of the grub was fixed while the actuators were filled to different pressures and a load cell was used to measure the resultant force from the link. The test rig and the force sensor in Figure 3.5 were rigidly mounted to a table and used to characterize τ.

Using the rig shown in Figure 3.5, a series of pressures were commanded to a singular bladder and the force on the load cell was recorded. The force measured from a singular actuator at given pressures and angles was plotted in Figure 3.6. Using the known link length, the torque contributions of the individual bladders and inherent joint stiffness was calculated as a function of pressure and angle.

The single bladder torque contribution seen in Figure 3.6 was used to identify a simple linear relationship between joint torque and pressure. θ was seen to have a linear influence on

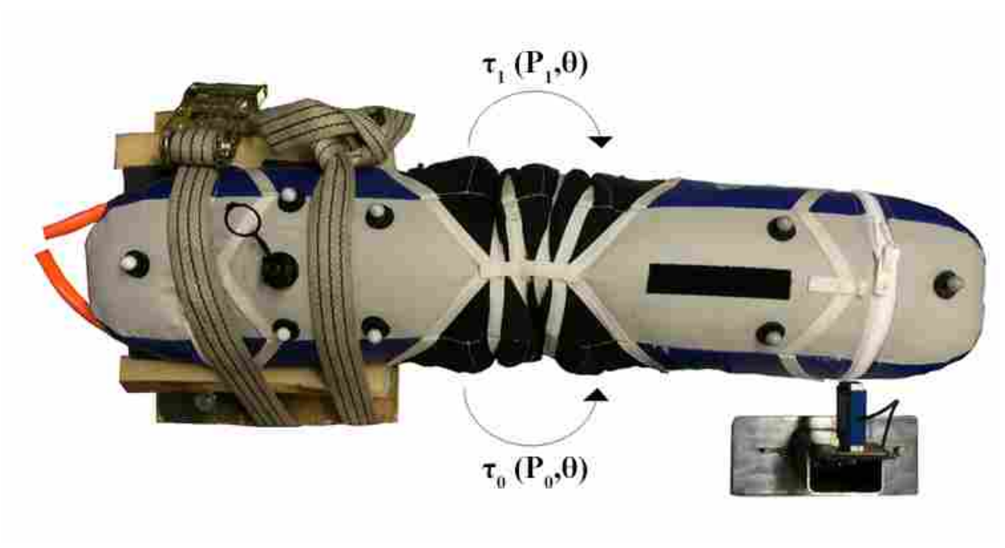

Figure 3.5: Testing setup with force sensor mounted at 0°

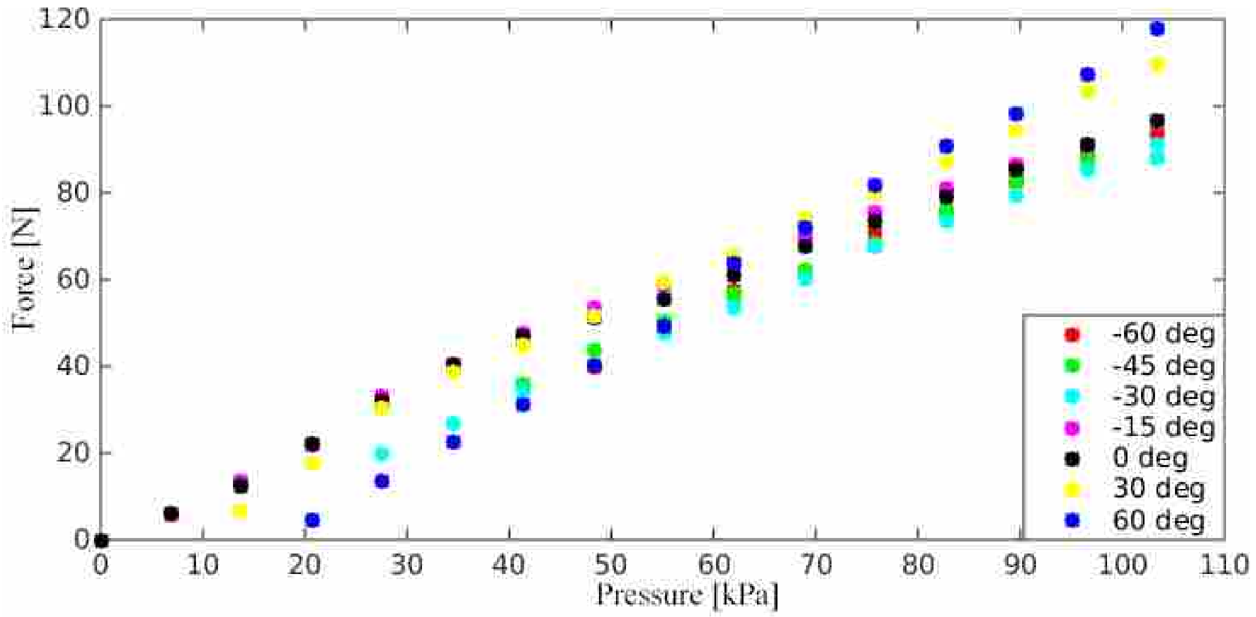

Figure 3.6: Actuator force at different pressures and angles

torque, indicating an inherent joint stiffness due to the fabric construction and pressurized internal bladders. Given these considerations, joint torque was calculated as:

$$\tau = \beta_0 P_0 - \beta_1 P_1 - K_s \theta \tag{3.14}$$

where β_0 and β_1 are torque coefficients for actuation bladders 0 and 1 respectively, and K_s joint stiffness.

While β_0 and β_1 were constant throughout the entire operational regime, through more experimentation K_s was found to be constant between ±15 degrees of 0 and increased linearly with theta for angles greater than 15 degrees.

$$K_s = max\left(K_{sm}, K_{sm} + m\left(|\theta| - 15\right)\right) \tag{3.15}$$

where K_s is joint stiffness, K_{sm} is a base stiffness value for the ±15 degree regime, and m is a constant scaling factor.

With this representation for torque, combined with the simplified link dynamics in Equation 3.5 and first-order pressure dynamics described in Equation 3.4, the dynamics were described in linear state-space form as:

$$\begin{bmatrix} \ddot{\theta} \\ \dot{\theta} \\ \dot{P}_0 \\ \dot{P}_1 \end{bmatrix} = A \begin{bmatrix} \dot{\theta} \\ \theta \\ P_0 \\ P_1 \end{bmatrix} + B \begin{bmatrix} P_{D,0} \\ P_{D,1} \end{bmatrix} \tag{3.16}$$

where:

$$A = \begin{bmatrix} \frac{-K_d}{I} & \frac{-K_s}{I} & \frac{\alpha_0}{I} & \frac{-\alpha_1}{I} \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -a & 0 \\ 0 & 0 & 0 & -a \end{bmatrix} \tag{3.17}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ b & 0 \\ 0 & b \end{bmatrix} \tag{3.18}$$

K_s is expressed as a constant coefficient rather than as a function of θ, as it is kept constant over the entire control horizon to conserve linearity, thus seen as constant in the state-space form.

3.4 Deep Neural Network Model

As described in Section 2.3, Deep Neural Networks (DNNs) can be used to approximate any non-linear function. A system with severe hysteresis and unknown state interaction, like those described in this work, is difficult to model even with non-linear dynamics. These difficult to model dynamics are a perfect candidate for universal function approximation with DNNs. A problem commonly associated with modeling dynamic systems entirely with neural networks, however, is stability, as discussed in [27, 28]. Any region outside the given set of training data carries no guarantees on performance or stability. In an attempt to assuage these concerns, the neural network in this work was instead used to estimate the pressure to theta map shown in Figure 3.1, to better describe actuator hysteresis and unmodeled dynamics. Instead of using the neural network to estimate an entire static map of θ, the network was used online to identify the coefficients describing the tangent planar surface derived in Equation 3.7 directly. These coefficients were calculated as $\frac{\delta\theta}{\delta P_0}$ and $\frac{\delta\theta}{\delta P_1}$ through finite differencing.

The following series of simplified equations were used to describe the calculations used with the constructed neural net;, for a more detailed description of the calculations including an example see Section 2.3.

A net with six inputs: $\dot{\theta}[k]$, $\theta[k]$, $P_0[k]$, $P_1[k]$, $P_{D,0}[k]$, and $P_{D,1}[k]$, and one output: $\theta[k+1]$, consisting of 4 layers with 200 nodes each, shown in Figure 3.7, was constructed using Google TensorFlow [29]. Google TensorFlow was selected over similar architectures for its simplified net construction process, which allowed for iterative design, high-speed graphics processor accelerated calculations, open source license, and quality documentation.

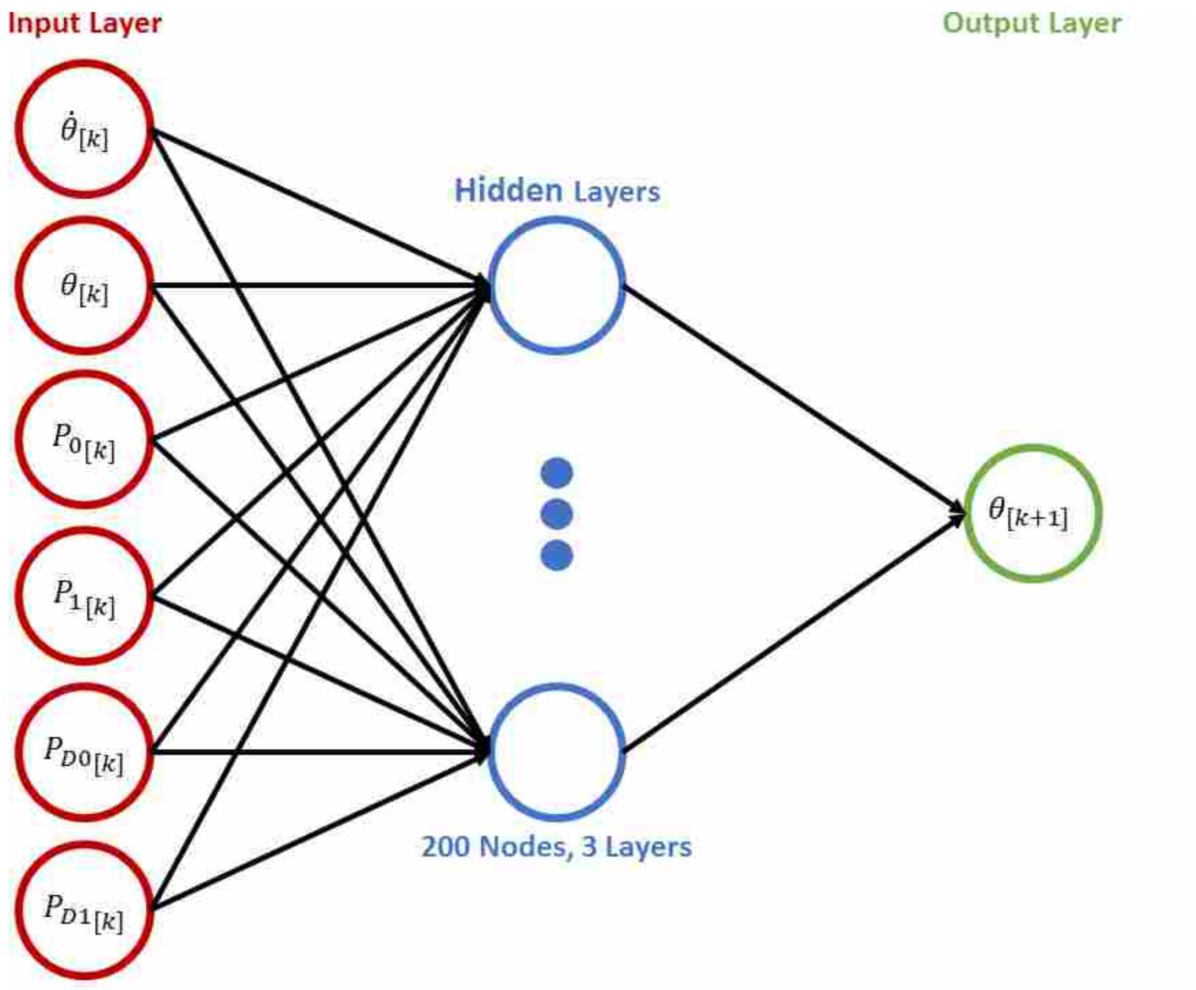

Figure 3.7: Neural network structure

The nodes in each layer were initialized using Xavier initialization for weights, a method developed by Xavier Glorot et al. in [30] for constructing large ANN matrices with roughly equal randomized gradients in all layers.

Between each layer, the inputs from the previous layer were multiplied by the matrix of node weights, W_H, and added to a vector of bias weights B_H. This node calculation was then run through a sigmoid function, serving as an activation threshold function to determine if the output of the node is on: $output = 1$ or off: $output = 0$:

$$output = \frac{1}{1 + e^{-W_H * input + B_H}} \tag{3.19}$$

where *input* is either the initial system inputs or the output vector from the previous layer and *output* the output of the current layer.

This calculation was repeated three times, until reaching the final layer, where the activation function was removed:

$$out = -W_H * in + B_H \tag{3.20}$$

The net described in Figure 3.7 takes in each state and inputs at the current timestep k and uses it to predict the value of θ at the next time step $k+1$. Full estimation of all states was not required for the desired structure of this model. Estimating full states or a state for multiple future timesteps would require a larger network with more inputs.

DNN Model: Training

Two sequences of step inputs were generated with randomized frequency and step amplitude for each actuator. Frequency for the steps was allowed to change with time independently between the inputs. This generated sequence of inputs was initially simulated using the torque model, described in Section 3.3, using the Matlab 2015b Lsim function. This step was used to verify the predictive capabilities of the constructed neural network.

Once satisfied with the DNN's predictive capabilities, the randomized trajectory of inputs were executed on the Grub platform and all states recorded. This data was then arranged into large arrays for feeding into the neural network, where the ANN inputs at k were organized as a vector:

$$ANN_{in}[k] = \begin{bmatrix} \dot{\theta}[k] \\ \theta[k] \\ P_0[k] \\ P_1[k] \\ P_{D,0}[k] \\ P_{D,1}[k] \end{bmatrix} \tag{3.21}$$

matched with a corresponding output:

$$ANN_{out}[k] = \theta[k+1] \tag{3.22}$$

This data set was randomly split into two separate groups, training data and validation data, at a ratio of 10-to-1 respectively. The training data was run through the neural network, resulting in a large vector of estimated $\theta[k+1]$. This estimated vector of $\theta[k+1]$ values was evaluated against the measured vector of $\theta[k+1]$ values. An error was calculated by subtracting the estimated vector from the measured vector and the resultant difference squared. This squared difference was then averaged across the entire data set. This final value was known as the Mean Squared Error (MSE). The Adam Optimizer, developed by Kingma et al. [31] and integrated into TensorFlow, was used to train the neural network, identifying the collection of node and bias weights that minimized the MSE.

At each iteration from the Adam Optimizer, the neural network was also evaluated with the validation data set and the validation MSE was compared to the training MSE. Throughout the optimization the MSE seen from both sets of data decreased together, but when the errors diverged significantly, the optimization procedure was stopped. This process was done to prevent overfitting of the training data to the DNN based on the methods described in [32], where if the optimizer continues to reduce MSE in the training data set the network will become less general.

Training was performing on an Ubuntu 14.04 workstation using the NVIDIA CUDA parallel computing platform and the CUDNN API on NVIDIA GTX 750 Ti and TitanX graphics processors.

DNN Model: Gradients

To calculate the desired $\frac{\delta\theta}{\delta P}$ values from the DNN, a function which only provided values for $\theta[k+1]$ given the current states and inputs, simple central finite differencing was used by adjusting the pressure states.

Google TensorFlow was designed to allow for offloading of computations to a graphics processor, which facilitates high-speed matrix calculations due to the architecture used in graphical processors. This resulted in no discernible loss in performance when calculating the output of the net for one vector of inputs, versus several vectors concatenated into an array. As a result, central differencing was seen to be reasonably efficient compared to both forward and backward differencing. The following delta vectors were used for central differencing:

$$D1 = \begin{bmatrix} 0 \\ 0 \\ \delta \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.23}$$

$$D2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \delta \\ 0 \\ 0 \end{bmatrix} \tag{3.24}$$

These delta vectors were combined with the input to the neural network at the current time step k and concatenated into a singular array:

$$\begin{bmatrix} ANN_{in}[k] + D1 \\ ANN_{in}[k] - D1 \\ ANN_{in}[k] + D2 \\ ANN_{in}[k] - D2 \end{bmatrix} \tag{3.25}$$

This array of inputs was fed into the neural network and a vector of outputs calculated:

$$\begin{bmatrix} ANN_{out+D1}[k] \\ ANN_{out-D1}[k] \\ ANN_{out+D2}[k] \\ ANN_{out-D2}[k] \end{bmatrix} \tag{3.26}$$

This array of outputs, each describing a $\theta[k+1]$ estimated with different inputs, was then used to calculate the partial derivative of θ:

$$\frac{\delta\theta}{\delta P}[k] \approx \frac{(ANN_{out+D}[k] - ANN_{out-D}[k])}{2\delta} \tag{3.27}$$

These partials were then fed into the state-space form described in Section 3.2:

$$\begin{bmatrix} \ddot{\theta} \\ \dot{\theta} \\ \dot{P}_0 \\ \dot{P}_1 \end{bmatrix} = A \begin{bmatrix} \dot{\theta} \\ \theta \\ P_0 \\ P_1 \end{bmatrix} + B \begin{bmatrix} P_{D,0} \\ P_{D,1} \end{bmatrix} \tag{3.28}$$

where:

$$A = \begin{bmatrix} \frac{-K_d}{I} & 0 & \frac{-K_s \frac{\delta\theta}{\delta P_0}}{I} & \frac{-K_s \frac{\delta\theta}{\delta P_1}}{I} \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -a & 0 \\ 0 & 0 & 0 & -a \end{bmatrix} \tag{3.29}$$

$$B = \begin{bmatrix} \frac{-K_s \frac{\delta\theta}{\delta P_0}}{I} & \frac{-K_s \frac{\delta\theta}{\delta P_1}}{I} \\ 0 & 0 \\ b & 0 \\ 0 & b \end{bmatrix} \tag{3.30}$$

While $\frac{\delta\theta}{\delta P_0}$ and $\frac{\delta\theta}{\delta P_1}$ are considered functions of the current states $X[k]$ and inputs $U[k]$, they are expressed as constants in state-space form since they are kept constant over the entire control horizon to conserve linearity.

3.5 Performance Comparison

Performance of each of the models was directly compared by simulating the models discretely over a fixed time horizon and comparing these open-loop simulations to known system measurements, seen in Figure 3.8. The models were tuned for performance over the entire operational range of the robot and Figure 3.8 only describes one such trajectory. As all models used identical pressure dynamics, pressure state estimation was not compared.

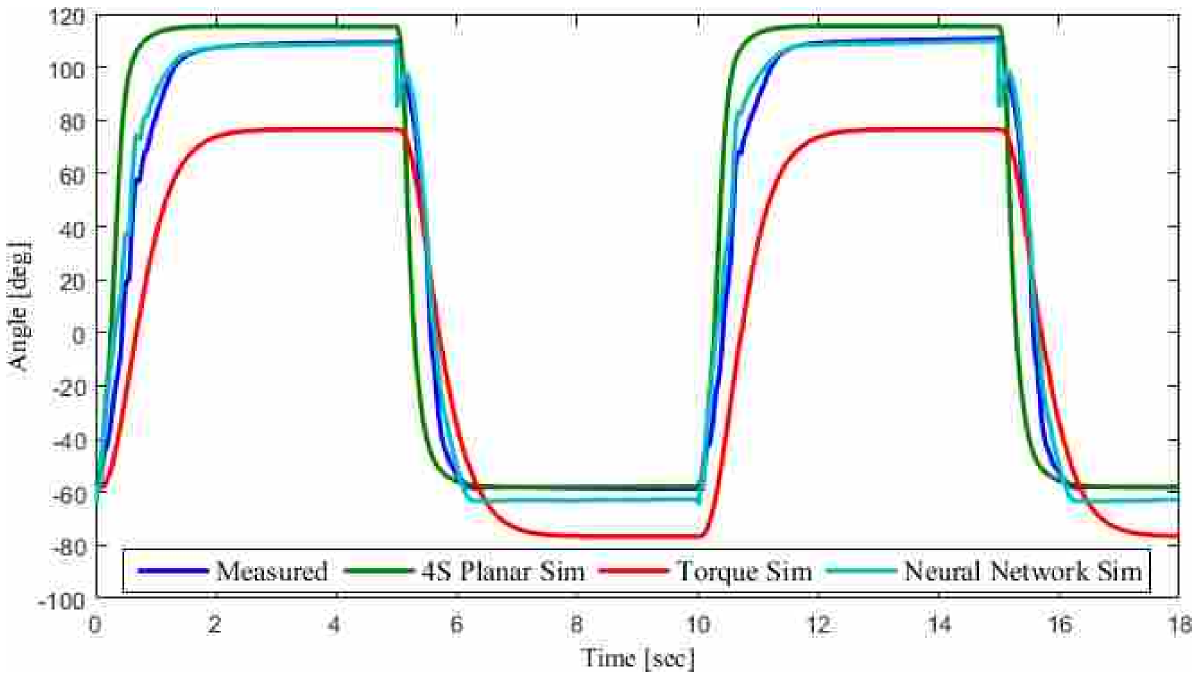

Figure 3.8: Predictive model capabilities - open loop

The planar impedance model estimated θ particularly well, as seen in Figure 3.8. While the model overshot the measured values slightly, prediction performance was particularly good. The model itself was set up to allow quick scaling of the θ to pressure mapping to any similar pneumatic joint. The map, seen in Figure 3.1, could be scaled to any range of pressures and joint limits. This model also allowed for a complete replacement of the θ map for any state relation surface.

The torque model was the simplest model created, as it used static values for nearly every dynamic parameter. This model was also very easy to simulate, as the dynamic parameters did not require any external calculations, but simulation often showed error at steady state, as seen in Figure 3.8.

The deep neural network model was the most accurate but it was also the most complex model, requiring significant hardware and external application programming interfaces (API). This model, however, was a complete black box, meaning true internal structure and workings were obscured, and required repeating the entire training procedure for any new platform.

While similar, each model represented significant research and changes to the dynamics. Their similarities allowed the same model-based controller, described in Chapter 4, to be used without significant alterations.

CHAPTER 4. SINGLE DOF MODEL PREDICTIVE CONTROL

In this chapter, a description of the controller used is first provided, followed by an analysis of control performance using each of the models described in Chapter 3 is compared to prior art described in [23].

4.1 Controller Description

Model predictive control (MPC) was applied using all models described in Chapter 3. MPC was used for its ability to handle model-based control, system constraints, and ease of iterative design. For a more detailed description of MPC, see Section 2.2.

A model predictive controller solves an optimization at every time step, simulating the predicted states over the horizon T by varying the inputs to produce the trajectory incurring the least cost subject to defined constraints. Throughout this work, the model predictive controllers were operated with a time horizon of $T = 20$ time steps at a rate of 300 Hz, effectively predicting 0.07 seconds into the future. With all models the discretized state-space matrices A_d and B_d, current states $\dot{\theta}[k]$, $\theta[k]$, $P_0[k]$, and, $P_1[k]$, previous inputs $P_{D,0}[k-1]$ and $P_{D,1}[k-1]$, goal angle θ_{goal}, target pressures $P_{T,0}$ and $P_{T,1}$, model constraints, and controller weights were fed into the MPC solver at every time step.

Pressure target values, $P_{T,0}$ and $P_{T,1}$, were low-weight cost value parameters that allow the setting of desired pressure operating point, which correlated to stiffness. The cost function minimized across the horizon T was:

$$minimize \sum_{k=0}^{T} \left(\|\theta_{goal} - \theta[k] + \theta_{int}\|_Q^2 + \|\dot{\theta}[k]\|_R^2 + \|P_0[k] - P_{T,0}\|_S^2 + \|P_1[k] - P_{T,1}\|_S^2 \right) \quad (4.1)$$

subject to the system model as constraints and the following additional constraints:

$$
\begin{aligned}
&X[k+1] = A_d X[k] + B_d U[k] \\
&|\theta| \leq \theta_{max} \\
&P_{min,i} \leq P_{D,i} \leq P_{max,i} \\
&|\Delta P_{D,i}| \leq \Delta P_{max,i} \\
&i = 0,1
\end{aligned}
\tag{4.2}
$$

where Q, R, S are scalar weights manually tuned for empirical performance, the same value of S is used for both pressure target costs, θ_{int} is an integrator term used to eliminate steady state error when needed, $X[k]$ is a vector of the system states at time step k, $U[k]$ is a vector of the system inputs at time step k, θ_{min} and θ_{max} are joint limits, P_{min} and P_{max} are pressure bladder limits, ΔP_D is the change in desired pressure from the previous time step and ΔP_{max} is the maximum change in desired pressure per time step permitted. Because simplified pressure dynamics were used, the slew rate restrictions serve to prevent valve chattering.

Target pressure, the P_T term in Equation 4.1, was introduced to the MPC cost function in an effort to improve controller performance. Initially the solutions produced by the MPC solver were often just above atmospheric pressure, as this solution allowed for the greatest perceived performance and lowest cost - low pressures allowed for high-ratio pressure differentials in a short period of time. While this solution agreed with the model, it was simply not physically true. System dynamics at extremely low pressures, 0-0.3 PSIg or less than 15 PSIa, were generally unknown and not tractable as the actuation bladders had not fully inflated. In an effort to combat this, a pressure target term was introduced to add a small cost for solutions close to atmospheric.

In order to effectively achieve angles away from the center position, different values of the pressure target, P_T, were sent to individual bladders. When bent to one side, the bladder on the inside of the bend must be at a lower pressure than the bladder on the outside of the bend. Individual pressure target values, P_T, are calculated based on the user set base stiffness pressure, P_S, according to the following equations:

$$
\begin{aligned}
P_T = M\left|\theta_{goal}\right| + P_S \\
\textbf{if } (\theta_{goal} \geq 0): \\
&P_{T,0} = P_S \\
&P_{T,1} = P_T \\
\textbf{else}: \\
&P_{T,0} = P_T \\
&P_{T,1} = P_S
\end{aligned}
\tag{4.3}
$$

where M is a constant which changes target pressure as a function of θ_{goal}, P_T is a calculated target pressure for the bladder on the inside edge of a bend and P_S is a target pressure applied to the bladder on the outside edge of the bend as well as the operating point for P_T function. P_S can be varied during operation from one actual time step to the next, but is constant over an entire MPC horizon.

An efficient solver was generated for the MPC problem using CVXGEN (see [22]), a web-based tool for developing convex optimization solvers. The optimization solver, written in C and subsequent Python code that called the solver, was run at 300 Hz. A predicted trajectory horizon of T = 20 time steps was used, a prediction of 0.067 seconds into the future.

Once solved, the first time step from the optimized trajectory of desired pressures was applied to the system and published over ROS. These desired pressures were received by the underlying pressure proportional-integral-derivative (PID) controller and valve position commands were then sent to the individual valves.

As described in Figure 4.1, the current pressure states were read and fed back into the pressure controller, while the angle states were fed into a Kalman filter. The Kalman filter was used to smooth out the angle estimation from the IMU, as described in Section 2.1.1. Both the current filtered angle states and measured pressure states were fed into the MPC controller along with user-specified θ_{goal} and P_S values.

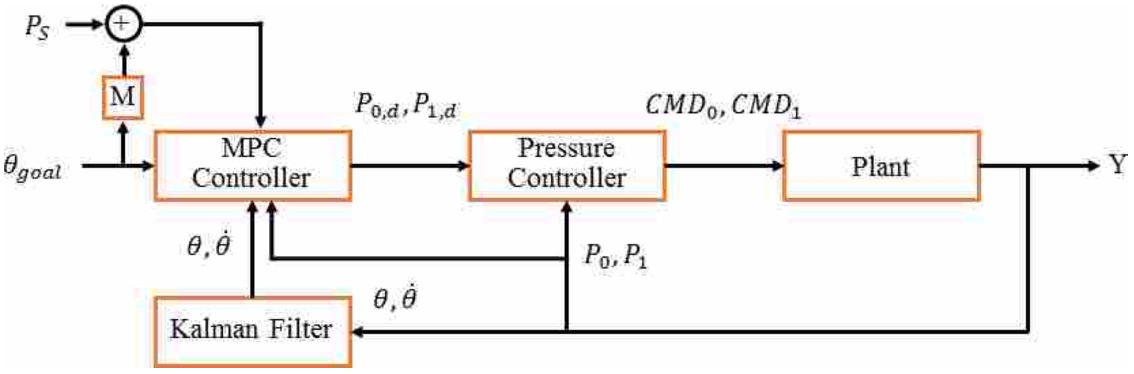

Figure 4.1: MPC control diagram for single-DoF joint and stiffness control

4.2 Performance

In this section the same controller described above was applied to each of the models detailed in Chapter 3. Each model was commanded to a series of step angle positions.

A series of 30-degree step angle commands ranging from -60 to 60, changing in increments of 10 seconds were commanded to each controller using a different model. The resultant angle θ over time was compared against the 2-state impedance model described in Section 3.1 using the model predictive controller developed in [26] commanded to the same angles. The resultant θ of both controllers and commanded θ_{goal} were plotted over time for comparison. The performance characteristics of each trajectory were also measured using an automated tool described in Chapter 6, where rise time, settling time, and percent overshoot were calculated for each step, averaged across the entire trajectory, and their results placed in a table.

4.2.1 4-State Planar Impedance Model

The controller within this section used the model described in Section 3.2, where tangent surfaces from Figure 3.4 are used to describe joint torque through an impedance model. This controller did not require usage of the integrator term θ_{int} in Equation 4.1.

Compared to the previous 2-state controller with only θ and $\dot{\theta}$, adding pressure states in the 4-state planar impedance controller significantly improved overall performance seen in Figure 4.2. As described in Table 4.1, the planar impedance model produced remarkably faster 90% rise time and 5% settling time while vastly decreasing percent overshoot. With angular step commands

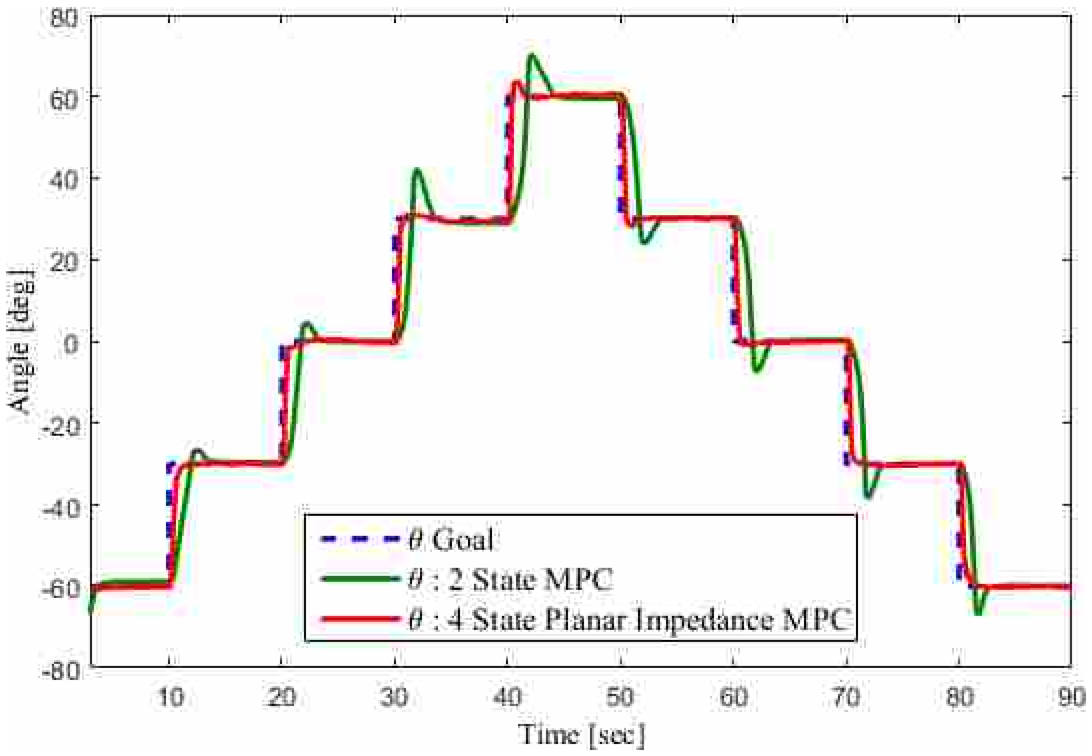

Figure 4.2: 4-State planar impedance model performance versus 2-state model

of only 30 degrees, the 2-state controller saw an average % overshoot of 24.408% or 7.3 degrees. The introduction of pressure dynamics reduced average % overshoot to just 3.711%, less than 1.12 degrees.

Table 4.1: Performance comparison between 2-state and 4-state planar impedance controllers

	Avg. Rise Time	Avg. Settling Time	Avg. % Overshoot
2-State MPC	1.191 sec	3.3156 sec	24.106%
4-State Planar Impedance	0.4552 sec	1.7785 sec	3.711%
Improvement	161.529%	86.428%	549.639%

4.2.2 Torque Model

The controller within this section used the model described in Section 3.3, where the individual contributions of actuator torque along with inherent joint stiffness are summed to describe joint torque.

Due to the differences in model structure, system performance with the torque model required usage of the θ_{int} integrator in the MPC cost function in Equation 4.1 to eliminate significant steady-state error. This difference was understood to be a result of the deviation from the impedance representation of torque towards a direct representation estimated from empirical data. The open-loop simulation, seen in Figure 3.8 showed significant steady-state error for θ using the direct torque representation.

In an effort to improve performance and mitigate integrator windup, a customizable integrator was created:

$$E[k] = \theta_{goal}[k] - \theta[k] \tag{4.4}$$

where E is angle tracking error at the current timestep k,

$$\dot{E}[k] = \frac{E[k]}{t[k] - t[k-1]} \tag{4.5}$$

where $\dot{E}$ is the time derivative of error and $t[k]$ is time at the current time step k.

$$\theta_{int}[k+1] = K_i E e^{-K_{es}|\dot{E}[k]|} + \theta_{int}[k] \tag{4.6}$$

where K_i is the integrator scale factor and K_{es} is the exponential scale factor.

This integrator was created under the idea that if $\dot{E}$, the time derivative of angle tracking error E, was high, indicating the arm was moving, the exponential function would prevent any buildup of the θ_{int} term. $\dot{E}$ was used instead of $\dot{\theta}$ to allow for integrator build up as the θ approached θ_{goal}. This custom integrator also provided two variables for tweaking integrator performance: K_i which controlled how quickly the integrator built up, and K_{es} which scaled the exponential function curve and determined how $\dot{E}$ the integrator built up whilst moving.

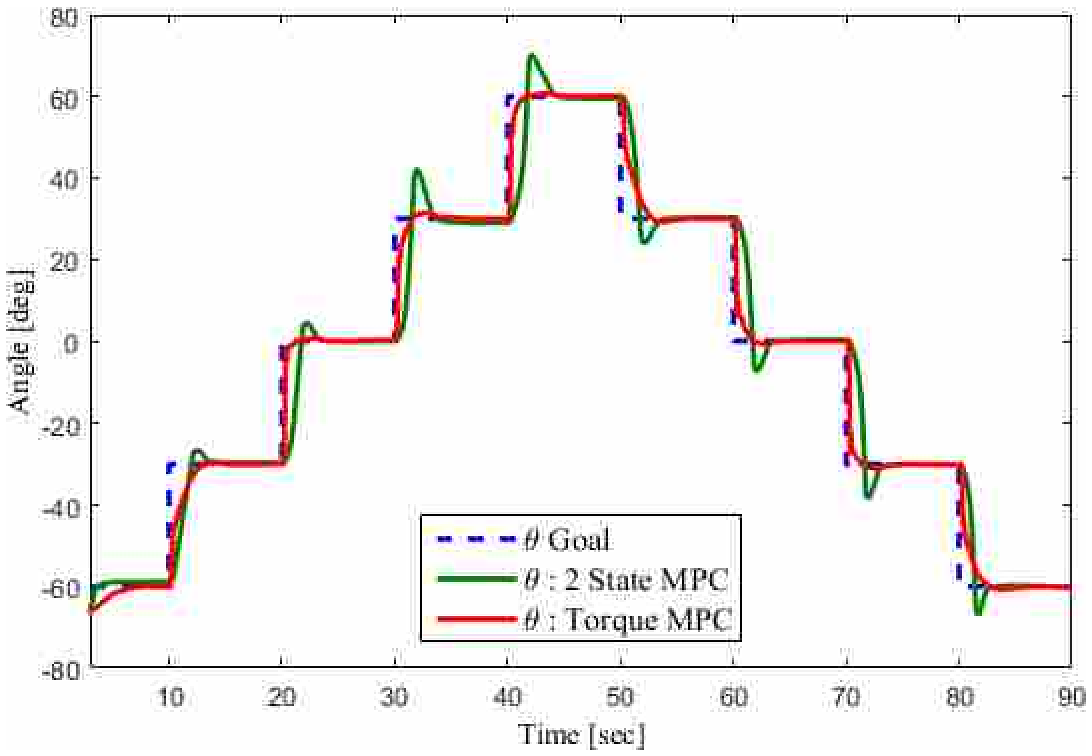

Figure 4.3: Torque model performance versus 2-state mode.

With the direct representation for torque, the controller performed significantly better than the 2-state MPC seen in Figure 4.3 and Table 4.2. The integrator was tuned to minimize percent overshoot which hurt 90% rise time compared to the planar impedance model.

Table 4.2: Performance comparison between 2-state and torque controllers

	Avg. Rise Time	Avg. Settling Time	Avg. % Overshoot
2-State MPC	1.191 sec	3.3156 sec	24.106%
4-State Torque-based MPC	1.320 sec	2.284 sec	2.610%
Improvement	-9.799%	45.192%	823.665%

4.2.3 Gradual Performance Degradation

The inflatable systems described in this thesis were notably difficult to model due to their inherent compliance, non-rigid structure, and hysteresis. These effects were mostly attributed to the general construction and assembly of the robotic platforms as they were all handmade. Over time the internal structure and actuation bladders would shift slightly, either due to usage or the process of reseating - deflation, removal, and reinsertion of internal inflatable bladders. This slight shift of the platform structure significantly altered unmodeled properties and dynamic performance.

The performance demonstrated in Sections 4.2.1 and 4.2.2 were, in great part, the result of significant hand tuning of the MPC cost function weights described in Equation 4.1 by the author of this thesis. This lengthy process of hand tuning required significant trial and error to identify weights which maximized desirable performance. Unfortunately there was no known formulaic process of calculating the proper control parameters like exists for traditional control methods.

The impact effect on performance caused over time was evident from significant degradation of previous controller performance on the same hardware that has been used regularly over a period of time. Running the controller described in Section 4.2.1 on the same Grub platform, described in Section 2.1.1, with identical source pressure and controller weights, after four months of regular usage demonstrated the significant performance degradation seen in Figure 4.4 and Table 4.3.

Table 4.3: Performance comparison between the 4-state planar impedance controller initially tuned and the same controller 4 months later.

	Avg. Rise Time	Avg. Settling Time	Avg. % Overshoot
4-State Planar Impedance	0.4552 sec	1.7785 sec	3.711%
4 Month Degradation	0.3642 sec	1.0288 sec	17.638%
Change	24.979%	72.866%	-78.963%

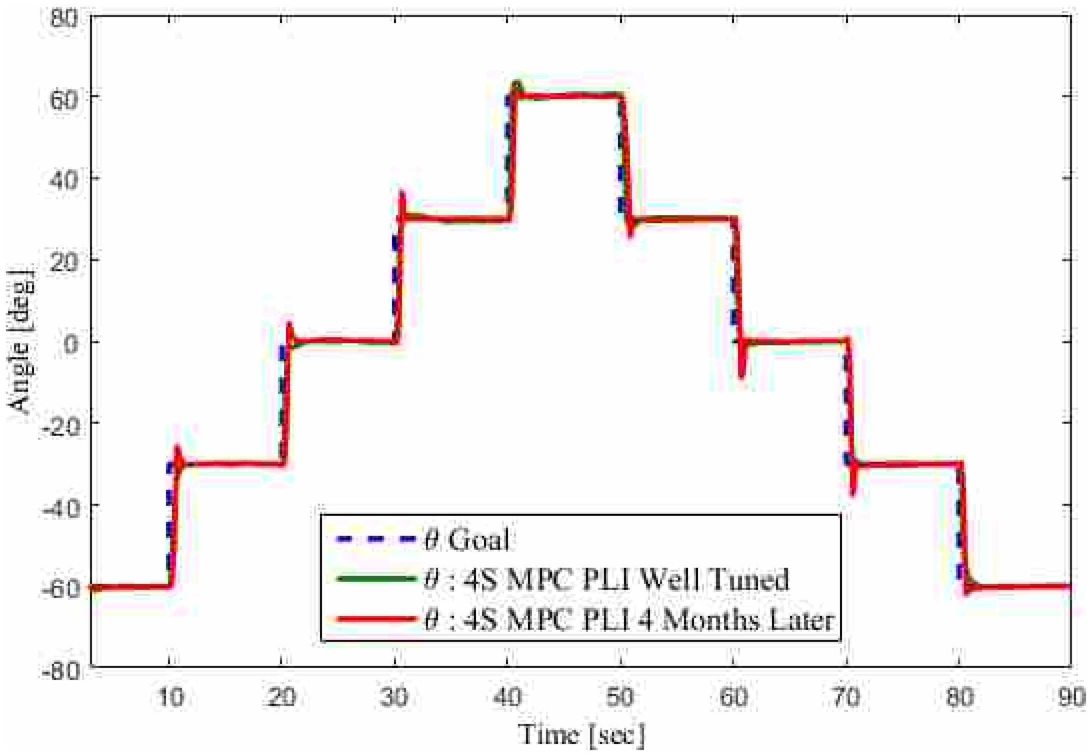

Figure 4.4: Negative performance impact due to plant changes

Significantly faster rise time, but also higher percent overshoot were seen in both Figure 4.4 and Table 4.3. It was likely that these significant changes in controller performance could be negated by significant retuning of the MPC weights or dynamic system parameters. This demonstrated a need for an automated procedure for the particularly lengthly process of tuning model predictive controllers. Preliminary work for an automated procedure was completed and is described later in this thesis in Chapter 6. The identification of performance changes were important to notate, as the controllers described in Sections 4.2 and 4.2.2 were tuned and benchmarked around the same time, whereas the controller described in 4.2.4 was tuned and benchmarked more than four months later.

4.2.4 DNN Model

The controller within this section used the model described in Section 3.4, where a deep neural network was used to estimate $\frac{\delta\theta}{\delta P_0}$ and $\frac{\delta\theta}{\delta P_1}$ for usage in an impedance model. This controller did not require usage of the integrator term θ_{int} in Equation 4.1.

While the neural network required running additional extra external processes, as described in Section 3.4, as it calculated $\frac{\delta\theta}{\delta P}$ using a separate graphics processor there was no impact upon MPC solve rate.

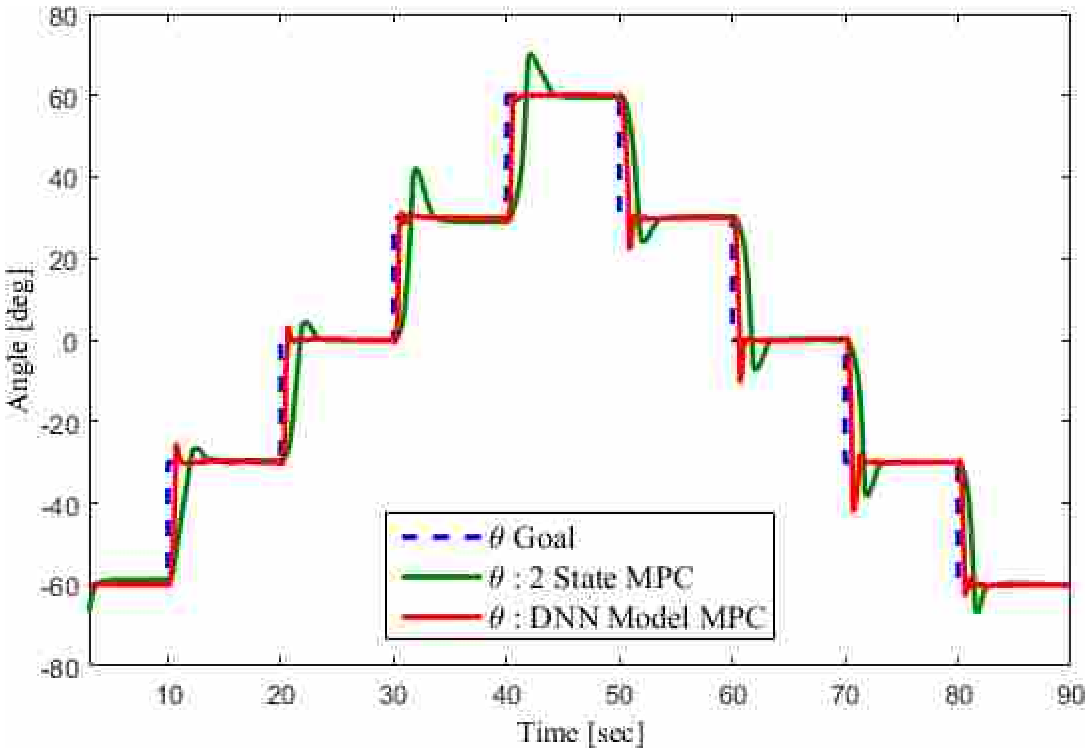

Figure 4.5: ANN model performance versus static map

While the DNN-based controller demonstrated improved rise time, as seen in Figure 4.5 and Table 4.4, percent overshoot was still a notable problem. The intent of this model was to potentially learn these significant unmodeled system parameters; however, performance was similar to that of the degraded planar impedance controller described in described in Section 4.2.3. The lengthy training process and black box nature of the DNN-based model made adjusting of dynamic parameters difficult and limited controller tuning abilities. Finite time resources resulted in a focus on simulated model accuracy over MPC performance, as the performance described in Section 4.2.1 required over 30 hours of hand tuning to achieve.

Table 4.4: Performance comparison between 2 State Impedance and ANN controllers

	Avg. Rise Time	Avg. Settling Time	Avg. % Overshoot
2-State MPC	1.191 sec	3.3156 sec	24.106%
4-State DNN MPC	0.376 sec	1.188 sec	18.174%
Improvement	216.588%	179.197%	32.636%

4.3 Variable Stiffness Tracking

In addition to improving controller performance, it was discovered that by varying the P_T values, one would effectively alter the joint stiffness during operation. The ability to adjust joint stiffness and joint angle simultaneously would allow these actuators to store energy, dampen unwanted oscillations, or adjust contact forces while still controlling position. While the exact relationship between pressure and stiffness was unknown for this platform, pneumatic spring stiffness equations were found in standard literature [33]:

$$K = \frac{nPA^2}{V} \tag{4.7}$$

where K is the pneumatic spring stiffness, n is the polytropic exponent, P is pressure behind the diaphragm or piston, A is the cross sectional area, and V is the volume of the fluid.

With the Grub platform seen in Figure 2.3, once bladders had filled the non-expanding envelope prevented additional changes in volumes when joint angle remained constant. During operation, n and A were also assumed to be constant. Through these assumptions, an increase in P should result in a direct linear increase in joint stiffness. Concurrent work was completed within the Robotics and Dynamics Laboratory at Brigham Young University to more accurately describe the joint stiffness of the robot platforms described in Section 2.1, however the focus of this thesis was the development of torque models and controllers.

In an effort to demonstrate this effect, the 4-state linear impedance model described in Section 3.2 along with the controller described in Section 4.1 was used to track θ_{goal} as well as various changes in P_S.

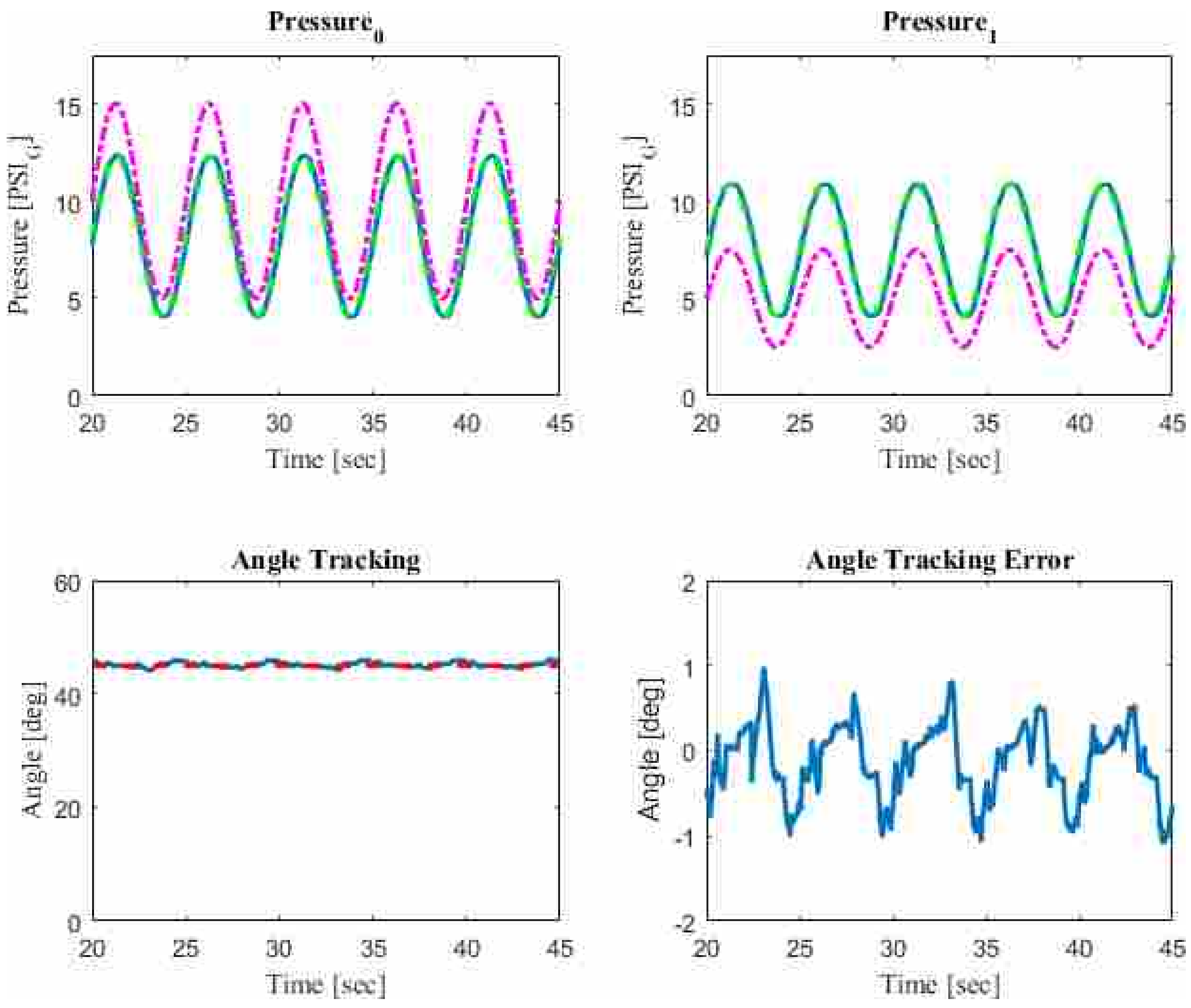

Figure 4.6: Results for Grub holding a constant angle with a sinusoidal pressure command which is related to stiffness

Given a constant θ_{goal} and a sinusoidal value for P_S, Figure 4.6 showed that the joint angle θ was maintained while changing effective joint stiffness. P_S was commanded as a sinusoid with a frequency of 0.2 Hz and an amplitude of 5 psig operating around 10 psig. Figure 4.6 demonstrates angle tracking error remained within 1 degree during changes of up to 200% in pressure in both bladders. As described in Equation 4.7, changes in pressure while maintaining a constant angle indicates a direct change in stiffness. This test demonstrated the controller's ability to alter P_S by as much 200% at a rate of 0.2 Hz while holding the joint angle relatively constant.

While the P_S value was varied sinusoidally, θ_{goal} step commands were also sent and followed. Figure 4.7 shows the resultant pressures and angle as θ_{goal} was stepped between 30 and -30

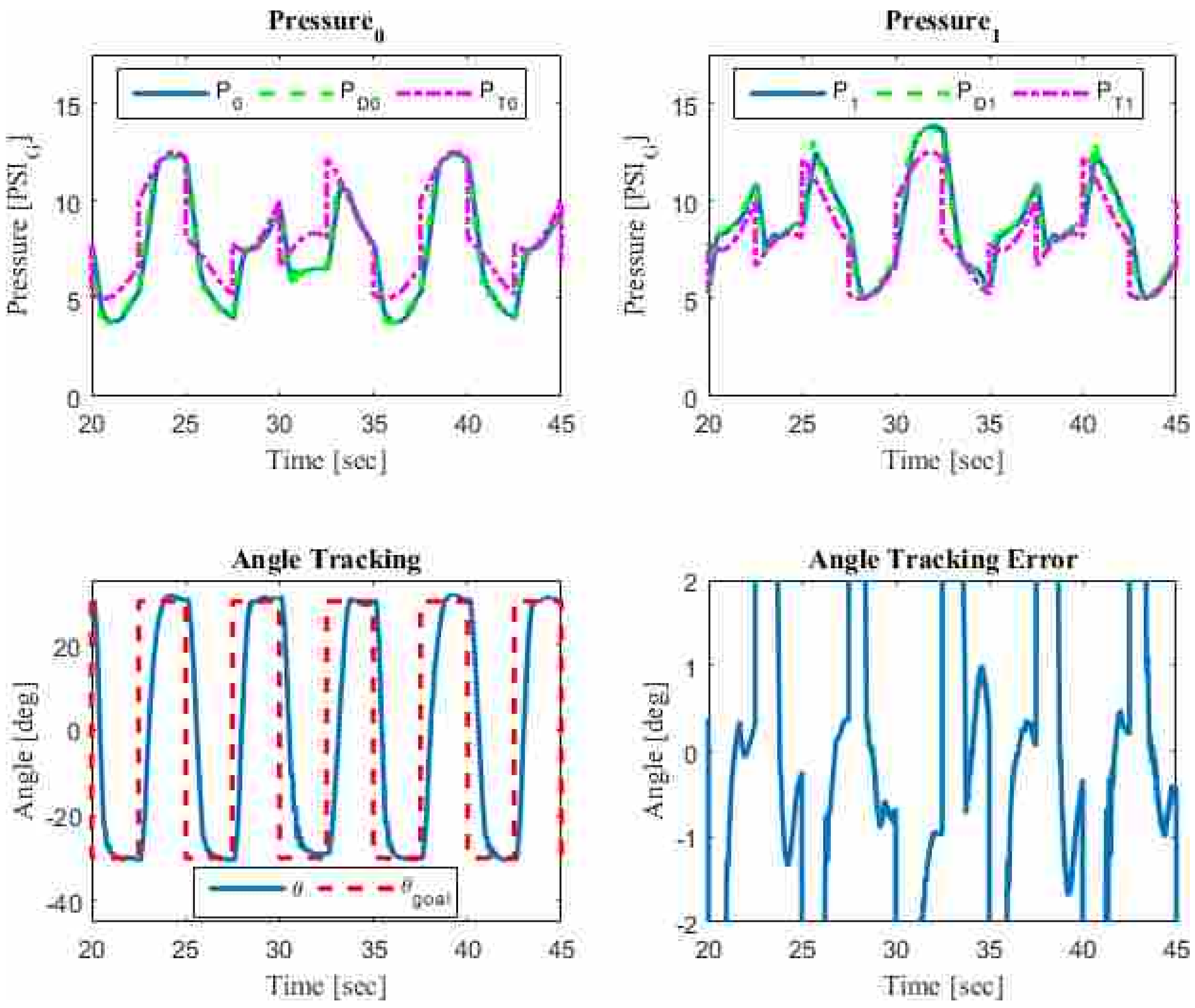

Figure 4.7: Results for a series of step commands in joint angle with a commanded sinusoidal change in pressure

deg at a rate of 0.2 Hz and the P_S as a sinusoid at 0.13 Hz with an amplitude of 2.5 psig operating around 10 psig. Once the commanded angle had been reached, tracking error remained less than 2 degrees while adjusting P_S. The large step swing in P_T values are a function of changes in θ_{goal} as described in Equation 4.3.

Figure 4.8 describes the same configuration as Figure 4.7 with the commanded frequencies reversed, where θ_{goal} was stepped between 30 and -30 deg at 0.13 Hz and the P_S a sinusoid at 0.2 Hz with an amplitude of 2.5 psig operating around 10 psig. As in the previous case, non-transient tracking error remained less than 2 degrees despite large changes in pressures.

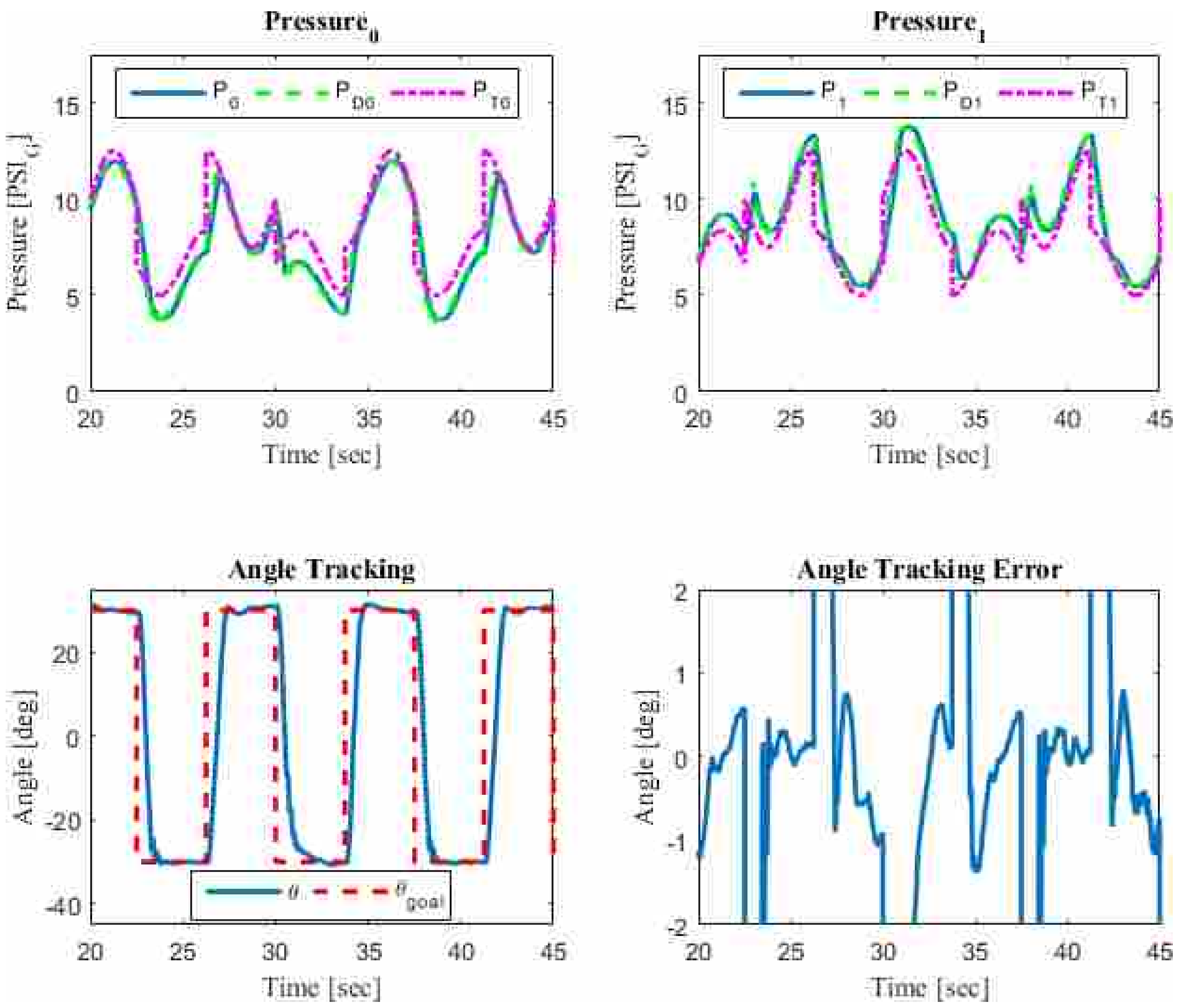

Figure 4.8: Same results as Figure 4.7 but with frequencies reversed

The tests shown in Figures 4.7 and 4.8 demonstrate the controller's ability to alter joint stiffness at variable rates while still achieving large changes in commanded angle.

4.4 Conclusions

While the DNN-based model was capable of learning many of the non-linear effects present in the system and showed very accurate open-loop state predictions in Figure 3.8, control performance in Section 4.2.4 was less than that of the well tuned 4-state planar impedance controller in Section 4.2.1 which performed best out of the three. The integrator used in the torque model controller, described in Section 4.2.2, performed very well. Mitigating steady-state error entirely, while avoiding significant overshoot or integrator windup.

When considering how applicable these controllers were to alternate systems, the 4-state planar impedance model controller appeared to be the easiest to apply as it would only require a straight forward relation surface from inputs to outputs and system specific adjustments to the dynamic model. Both the torque and DNN models would require extensive testing or training to achieve similar performance.

The addition of the ability to adjust joint stiffness to all model predictive controllers within this work, detailed within Section 4.3, added significant utility to this control methodology. While applications were not explored within this thesis, the simple implementation sets the stage for use in future research.

CHAPTER 5. EXPANDING CONTROLLERS TO MULTI-DOF

This chapter describes the implementation of a model predictive controller, described in chapter 4, on the multi-DoF King Louie platform described in section 2.1.2. This chapter is specifically focused on control of the first two joints of the right arm seen in Figure 2.4 as Joint 0 and Joint 1. These joints were selected as they differed significantly from the Grub joint in actuator size and joint orientation.

5.1 Sensing and Control

Each of the joints on the right arm of the King Louie platform were treated as uncoupled and followed the same dynamics as the single-DoF Grub platform described in Section 2.1.1. Each joint was considered independent of the rest. Although both gravitational and inertial effects were more prominent on the King Louie platform compared to the Grub, the model predictive controller used was robust against disturbances and modeling errors. This minimal impact on performance was believed to be a combination of several platform specific factors. The lightweight design of the platform resulted in the entire left arm, structure, sensors, and actuation, weighing less than 10 pounds total. The lightweight structure of this arm was actuated by high-pressure bladders, operating as high as 20 PSIg, resulting in high forces on the internal structure. These forces were expected to be significantly higher than any inertial or gravitational effects, resulting in pressure control having the greatest influence upon system dynamics. The influence of these effects on inflatable systems was an open area of research, but control performance expressed in this work was more than acceptable as an initial exploration into the feasibility of controlling multi-DoF soft, pneumatically actuated platforms.

The angle controller used on King Louie was the exact same model predictive controller described in Section 4.1. One controller was started for each joint, which operated independently of all other joints. A few special considerations were made for each model beyond significant tuning of the MPC weight values Q, R, S from Equation 4.1 for each controller.

As described in Section 2.1.2, the joint angles on the King Louie platform were not measured with traditional rotary positional sensors such as encoders or potentiometers. These joint angles were instead estimated based on measured link orientation and known joint kinematic relations. Additional details regarding this process can be found in Section 2.1.2.

5.2 Model Specific Considerations

As discussed previously in Section 4.4, certain models required additional consideration when adapting to a different platform and expanding the controller to multiple degrees of freedom. This section serves to describe any changes that were required for the model or controller.

5.2.1 4-State Planar Impedance Model

For the model described in section 3.2, controllable joint limits were measured as the maximum angle θ the joint could possibly achieve at maximum pressure in one actuation bladder, 20 psig, with atmospheric pressure in the antagonistic bladder, 0 psig. The values, θ_{min} and θ_{max}, were fed into the MPC controller as constraints. These values were also used to scale the θ to pressure map from figure 3.1. The minimum and maximum angles from the pre-existing map were extracted as θmap_{min} and θmap_{max} and were used to calculate a scaling factor:

$$Scl = \frac{\theta_{max} - \theta_{min}}{\theta map_{max} - \theta map_{min}} \tag{5.1}$$

Using this scaling factor, the 2D array describing the θ map was scaled all at once during controller initialization:

$$scaled\theta_{map} = Scl * \theta_{map} + \theta_{min} - Scl * \theta map_{min} \tag{5.2}$$

This scaling factor allowed the map of θ to pressure to be scaled for each joint, which varied in both size and joint limits.

5.2.2 Torque Model

The model described in Section 3.3 only required fitting the β torque coefficients in Equation 3.14 to collected data. State information was collected for a series of steps in desired pressures. Torque was then simulated using this data, and the β values were tuned by hand to best fit the simulated data to the measured data. This process was completed for both shoulder joints.

5.2.3 DNN Model

For the model described in Section 3.4 an entirely new network needed to be constructed. Rather than operating two separate neural networks, it was decided that the states between the joints were sufficiently coupled to instead operate one neural network with more inputs compared to what was produced in Section 3.4. While gravitational and inertial effects were ignored in the model dynamics, it was believed that the DNN could potentially learn and express these effects through $\frac{\delta\theta}{\delta P_0}$ and $\frac{\delta\theta}{\delta P_1}$.

The internal neural network structure remained the same as in Section 3.4 with 4 layers and 200 nodes per layer; however, the number of inputs was increased to 12 and outputs to 2 where:

$$ANN_{in}[k] = \begin{bmatrix} X_0[k] \\ U_0[k] \\ X_1[k] \\ U_1[k] \end{bmatrix} \tag{5.3}$$

matched with a corresponding output:

$$ANN_{out}[k] = \begin{bmatrix} \theta_0[k+1] \\ \theta_1[k+1] \end{bmatrix} \tag{5.4}$$

where $X_n[k]$ represents the states: $\dot{\theta}$, θ, P_0, and P_1, of joint n at time step k, $U_n[k]$ represents the inputs: $P_{D,0}$ and $P_{D,1}$ of joint n at time step k, and $\theta[n]$ represents the joint angle of the joint n.

The training process used was identical to that which was described in Section 3.4, but instead four randomized pressure input step trajectories were generated and applied to both joints simultaneously.

5.3 Performance

In order to compare controller performance directly to the 2-state impedance model derived in Section 3.1, the test case described in [25] was used. This test case involved moving the arm between two static poses seen in Figure 5.3 through a step input, where Joint 0 was actuated between -5 and -25 degrees, while Joint 1 was actuated between 0 and 60 degrees. The distal joints were ignored for these trials due to hardware limitations from leaks and computational limitations in model scaling - where the Neural Network model could only solve on immediately available graphics processors at the needed rates when configured to the described size in Section 3.4.

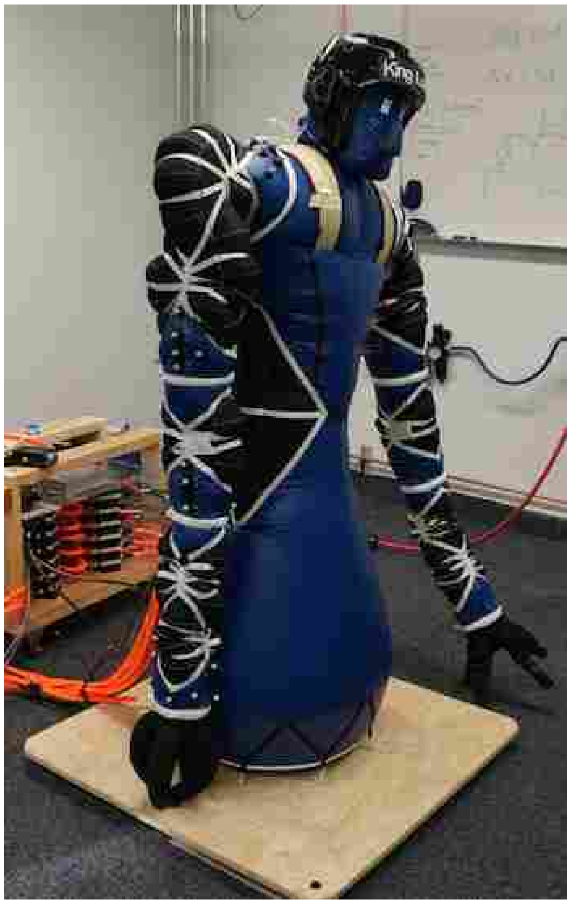
Figure 5.1: King Louie initial pose

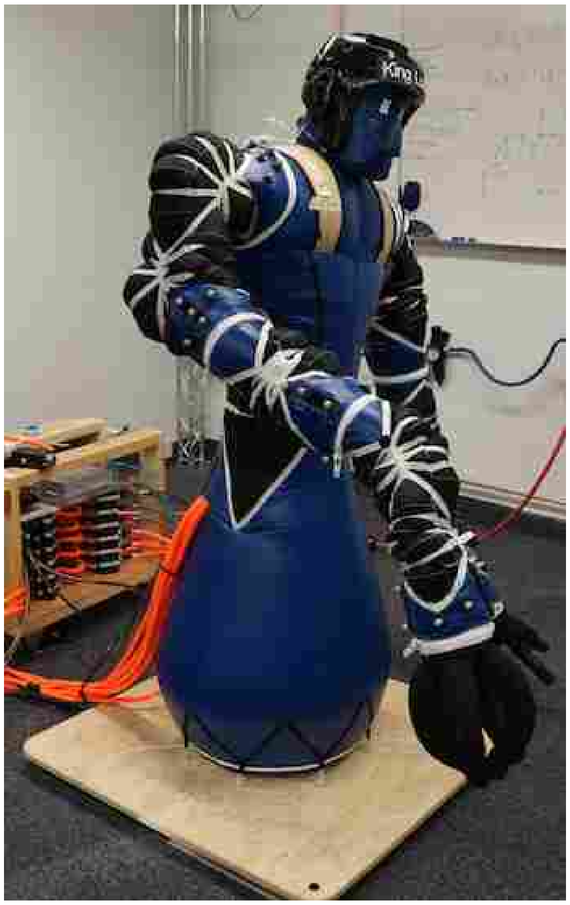
Figure 5.2: King Louie final pose

Desired angles for testing were commanded to controllers for all four models described in Chapter 3, where at the start the arm was completely drained, with all actuators at 0 PSIg. Joint 0 and Joint 1 were commanded to -5 and 0 degrees, respectively upon starting the controller. After

15 seconds, Joint 0 and Joint 1 were commanded to -25 and 60 degrees, respectively. All states were recorded from 0 to 30 seconds. The resultant joint angles were plotted together in one graph, seen in Figure 5.3, rather than separately.

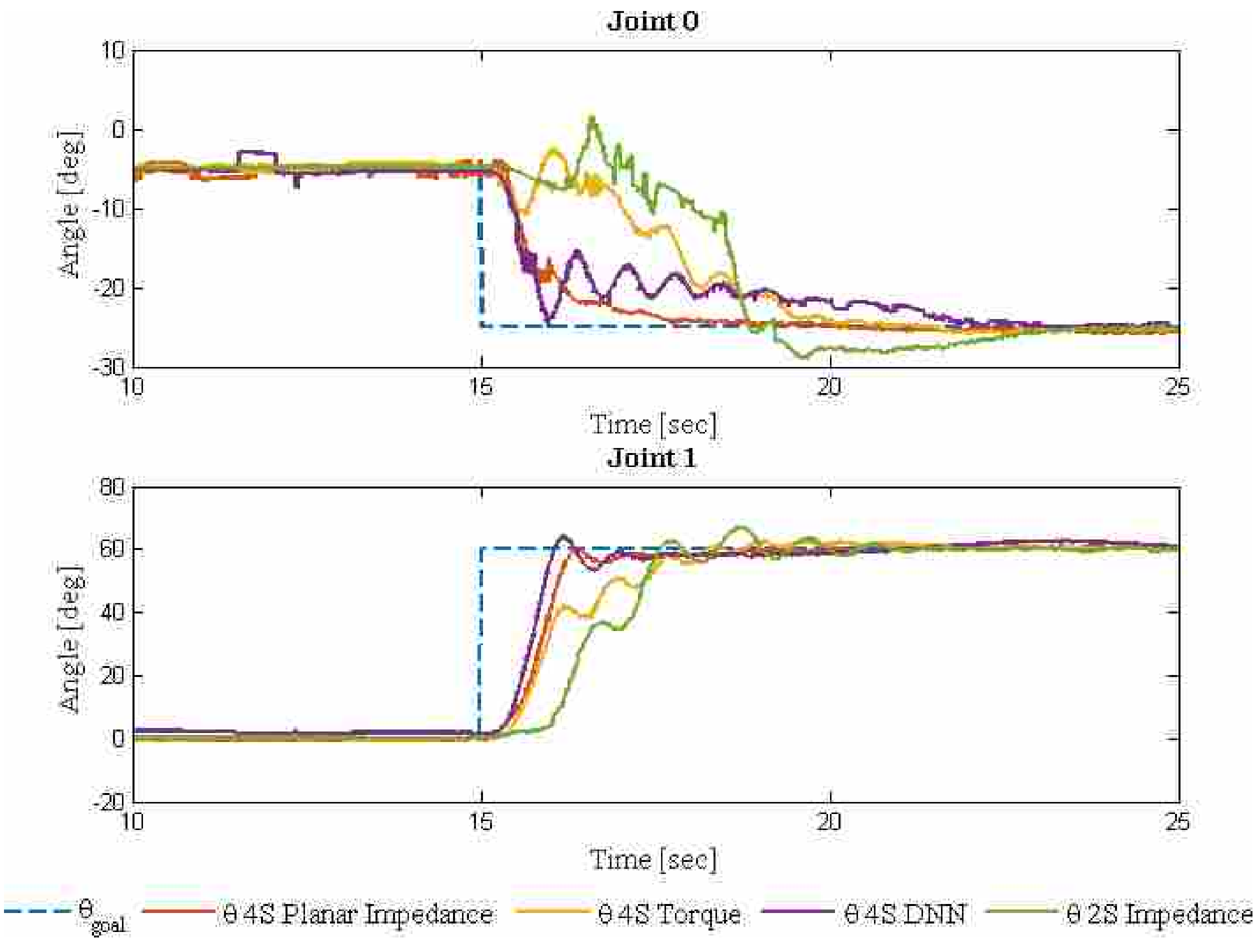

Figure 5.3: Control performance using MPC on various models

As seen in Figure 5.3, the controller using the 4-state planar impedance model outperformed the others once again, as it achieved fast rise time with limited oscillations, and minimal overshoot in both joints. While the torque based model controller still performed better than the 2-state controller, it exhibited a number of oscillations which negatively impacted settling time in both joints. The DNN model controller performed quite well in Joint 1, arguably as good as the planar impedance model. However, oscillations in control of Joint 0 were common with the DNN based model controller, this was believed to be due to the neural network training process. This objective function may have have prioritized the proportionally larger error seen from the wider operational range in Joint 1. Future work could potentially mitigate this by constructing separate networks for each joint or normalizing the relative angle prediction error to the joint specific operational range.

5.4 Conclusions

The control performance seen in Figure 5.3 demonstrated the great difficulty in achieving desired joint angles with fully inflatable, multi-DoF soft robots. While the 4-state planar impedance model performed best, it was believed that control performance could still be improved with the DNN-based controller. For improved system performance and expansion to even more joints, the elimination of gravitational and inertial effects should be reconsidered and better suited DNN models should be used.

CHAPTER 6. AUTOMATED MPC TUNING PROGRAM

As described in Chapter 4, the model predictive controller used has multiple weights in the optimized cost function that can be adjusted which significantly impact system performance. While various methods have been explored to guide the tuning process of these types of controllers in [34, 35], no standard automated procedure existed. Even with these guides, the actual process of tuning was still done predominately by hand, which is quite difficult with multiple control parameters. This chapter details the process of developing an automated model predictive control tuning tool and demonstrates its function through control improvements in two test cases.

6.1 Automated Performance Metrics

Determining the quality of control performance from a plot can be completed with relative ease visually by a skilled operator. Having a computer quantify this performance automatically with discrete data sets was not as straight forward, but many tools and standards already existed to do this. Several commonly used performance metrics as defined in [36] were selected for this work: rise time, percent overshoot, settling time, steady-state error, and oscillatory behavior. In order to use these metrics in an autonomous optimization process, these metrics needed to be measured automatically.

The model predictive controllers within this work sought to control θ to θ_{goal} and as such all metrics were calculated using these values over time as well as the time vector T.

In order to perform these calculations automatically, it was necessary to first know when the step was initiated T_{step} and second, the exact data index match where the step input was commanded. This process helped identify the exact data point in the discrete vector of time T closest to the step through the following equation:

$$\begin{aligned}[val, id_{start}] = & \underset{k}{min}(|T_{step} - T[k]|)\end{aligned} \tag{6.1}$$

where *min* is a minimization function which accepts vectors and returns both the value and index, *val* is the value of the minimization id_{start} is the index k which is closest to the commanded time. This id_{start} value is used to identify all measured states the start of the commanded step.

The value of $\theta[id_{start}]$ was recorded as the angle at start of the step, the 0% value. The step size, θ_{step}, was calculated as the difference between the $\theta[id_{start}]$ and $\theta_{goal}[id_{start}]$, as id_{start} was the index when the step was initiated.

Percent overshoot was defined as a signal exceeding its intended target. Utilizing the index of the commanded step id_{start} as a starting point, the maximum absolute value of θ before the next commanded step was identified as the largest overshoot. If the largest value of θ was greater than the θ_{goal} percent overshoot was calculated as:

$$P_{ovr} = \frac{|max(\theta) - \theta_{goal}|}{\theta_{step}} * 100 \tag{6.2}$$

If no value in θ was greater than θ_{goal} within the selected region, percent overshoot was claimed as 0.

Rise time was defined as the time required for the control response to rise from x% to y% of its final value in [36]. For the purpose of this work, 90% rise time was considered, which was defined as the time required to move between 5% and 95% of the desired value.

The time indexes corresponding to 5% and 95% of the step were calculated as:

$$[value, idn\%] = \underset{k}{min}(|\theta[id_{start}] + n\%\theta_{step} - \theta[k]|) \tag{6.3}$$

where *idn%* is the index which described where θ was closest to *n%* of the step. This equation was used to calculate the indices corresponding to both 5% and 95% of the step.

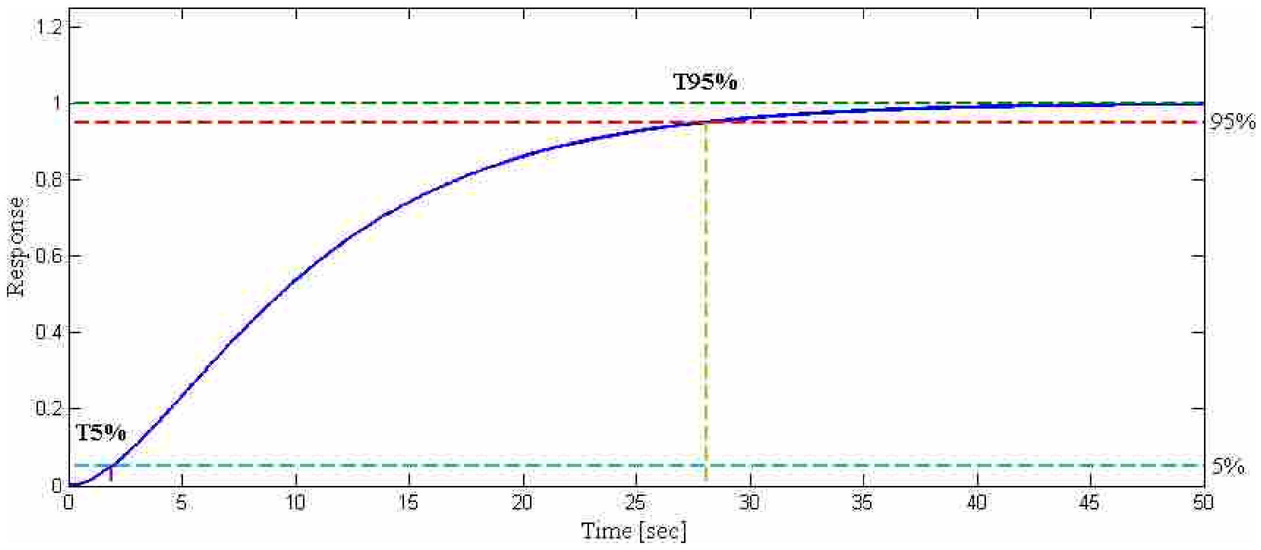

Figure 6.1: Illustration of rise time calculations

With these indexes, the rise time was calculated in seconds as:

$$T_{rise} = T[id95\%] - T[id5\%] \tag{6.4}$$

The illustration in Figure 6.1 was used to describe this identification of indices visually. The minimum function in Equation 6.3 was bounded by the index of the initial peak found for percent overshoot. This was done to avoid capturing oscillations which traveled through the 95% region.

Settling time was defined as the time between the initial commanded step and when the commanded state remained within a defined error band around the final value; this is illustrated in Figure 6.2.

The final value could be either identified as the value of θ at steady-state value or θ_{goal}. For the purpose of this work, the steady-state value θ represented as θ_{ss} was chosen, as not all controller solutions were able to approach θ_{goal} within the defined band. A 5% band was selected for this work, indicating the controller must keep θ between $\pm 2.5\%$ of θ_{step} around θ_{ss}. The steady-state value was calculated as the average θ over the last 50 time steps before a new θ_{goal} was commanded. The $\pm 2.5\%$ values were calculated and used as an errorband. Starting at the last index before a change in θ_{goal}, values of $\theta_[k]$ were decremented and compared against the error band. If a value of θ was found to be outside the error band, the index of the previous θ value

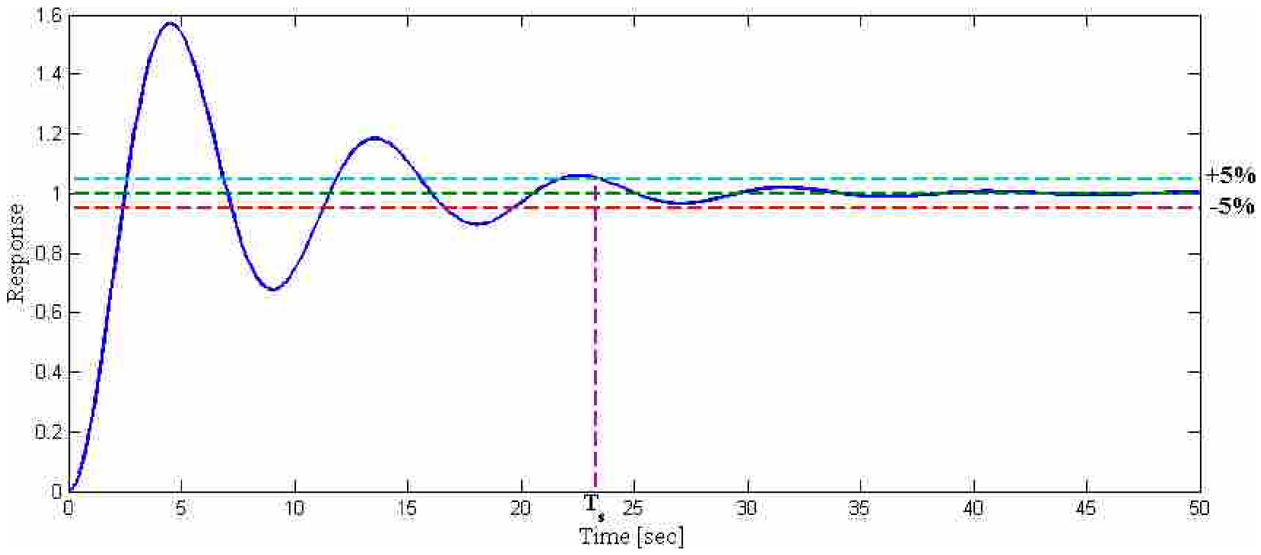

Figure 6.2: Illustration of settling time calculations

was recorded as id_{ss}. The time vector T was then used to define settling time using the following formula:

$$T_{set} = T_[id_{ss}] - T_[id_{start}] \tag{6.5}$$

Error steady-state was defined simply as :

$$E_{ss} = |\theta_{ss} - \theta_{goal}[id_{start}]| \tag{6.6}$$

Oscillatory behavior was difficult to quantify, as minor oscillations were acceptable if they were slow and small, but not if they were large and fast. A straightforward way to identify these peaks for discrete data was identified as finding zeros of $\dot{\theta}$ that exceed a certain slope threshold across the X-axis. This empirically selected threshold ignored slow waffling close to the axis or minor jitters in data; only large oscillations were recorded and marked with an asterisk, as seen in Figure 6.3.

The formula below was used to identify oscillations in θ:
where id_{start} is the index of the initial θ_{goal} command, $idxend$ is the last index before a change in θ_{goal}, $idxlist$ is a list of indices where oscillation peaks occur, and $Threshold$ is a set limit for a

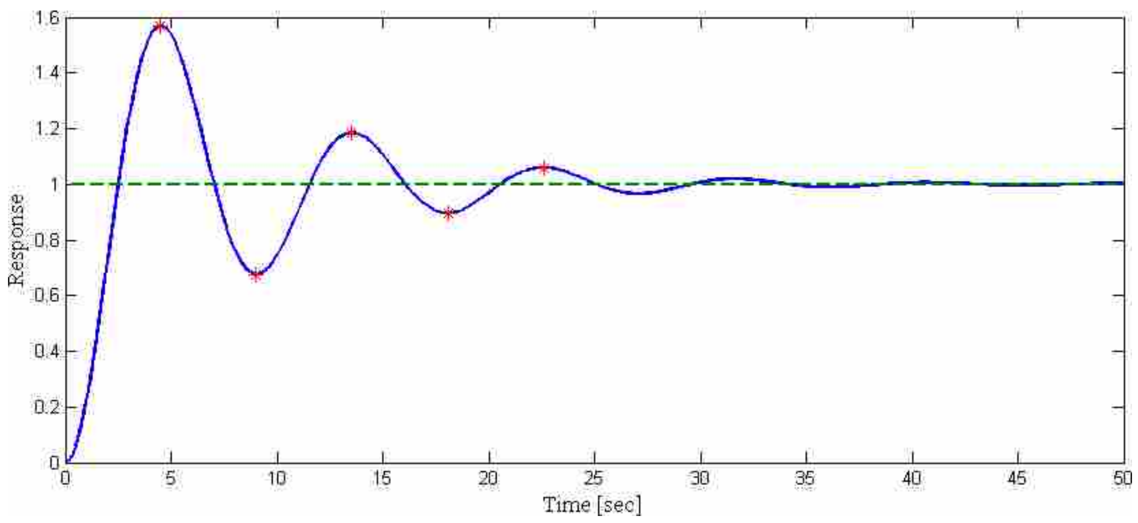

Figure 6.3: Illustration of oscillation identification

```
**for** $k = id_{start}$ to $id_{end} - 1$ **do**
    **if** $\dot{\theta}_{[k]} < 0 \quad \& \quad \dot{\theta}_{[k+1]} > 0$ or $\dot{\theta}_{[k]} > 0 \quad \& \quad \dot{\theta}_{[k+1]} < 0$ **then**
        **if** $|\frac{\dot{\theta}_{[k]} - \dot{\theta}_{[k+1]}}{T_s}| > Threshold$ **then** $idxlist = [idxlist, k]$
        **end if**
    **end if**
**end for**
```

minimum derivative that should produce an oscillation index. The number of oscillations $OSC_{\#}$ was determined by the length of the $idxlist$.

6.2 MPC Tuning

Using modified versions of the methods seen in [35], these optimization techniques were applied to the 4-state planar impedance controller on the Grub described in Section 3.2. An objective function was constructed using the derived numerical approximations of performance metrics:

$$min \sum |T_{rise}|_D + |P_{ovr}|_E + |T_{set}|_F + |E_{ss}|_G + |Osc_{\#}|_H \tag{6.7}$$

where D, E, F, G, H are weights for each individual metric based on what is most important to the user.

The MPC cost function weights were combined into a single single vector W:

$$W = \begin{bmatrix} Q \\ R \\ S \end{bmatrix} \tag{6.8}$$

where Q, R, and S are the MPC cost function weights described in Equation 4.1.

The Matlab gradient-free optimization function $Fminsearch$ was used to calculate a W to enter into the objective function. This objective function would write these weights to a file and then waited until the physical Grub platform completed the predetermined trial. The model predictive controller waited until the file containing weights was written by the objective function, loaded these weights, and the weights were sent to the controller along with a defined angle trajectory to track. After completing the defined trajectory, the controller was restarted and all logged states were saved to a .mat file. This .mat file was then imported into Matlab by the objective function and the automated metric identification was performed. Equation 6.7 was used to calculate the controller performance and a singular objective value was returned to $Fminsearch$. This process continued until convergence criteria were met or the optimization was manually ended.

6.3 Tuning Results

As shown in section 4.2.3 small changes to a dynamic model resulted in significantly different MPC performance. To observe these effects empirically, two additional sets of dynamic model parameters were generated for the 4-state planar impedance model described in Section 3.2 by adding significant disturbances to specific system dynamic parameters and using these disturbed models within the same model predictive controller. The first set of parameters increased both system inertia I and K_d by 50%. The second set of parameters increased K_s by 50% and reduced K_d by 50%.

As seen in Figure 6.4 both parameter sets with disturbances demonstrated relatively poor control performance when using the controller tuned for the original model. By using the automated tuning procedure described in section 6.2, performance was significantly improved and brought closer to the hand tuned set described in [24].

The controller weights used for initial parameter set were the same as those used in [24], $W = 6.5$, $R = 0.01$, $S = 0.0005$. These weights were used as the initial values provided to

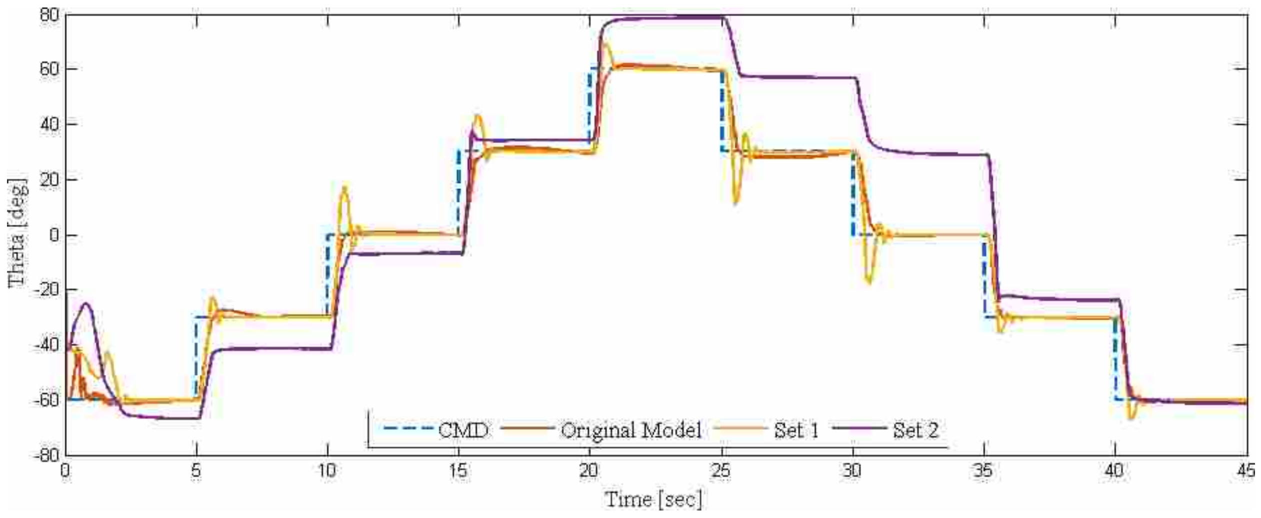

Figure 6.4: Comparison of untuned controller performance

fminsearch for both dynamic parameter sets. The system was allowed to optimize until desirable results were found.

On the first set of altered dynamic parameters, significant performance increases were found relatively quickly as seen in Figure 6.5. After only 10 iterations, the output of the objective function was decreased by over 60% of the original value.

The solver was stopped after 97 iterations as desired performance was achieved. Steady-state error was largely unaffected, as the initial parameters provided adequate performance. Percent overshoot, however, was reduced from 59.55% to 2.38%, without any impact upon rise or settling time. The number of oscillations was reduced from an average of 4 per step to 0.125. This was a significant improvement in performance all around, which brought control performance with a notably different set of dynamic parameters on par with the original model.

On the second set of dynamic parameters, the most significant improvement was on steady-state error. Steady-state error was reduced from an average of 13.5 degrees per step to just 1.03 degrees, as seen in Figure 6.6. The optimization attempted multiple times to set the weight on pressure target S to a negative value, causing instabilities. These instabilities can be seen as spikes on the iteration cost plot.

While the tuned performance was improved dramatically, the tuning introduced a large number of oscillations, averaging five per step. This can be expected, as both system I and K_d were increased.

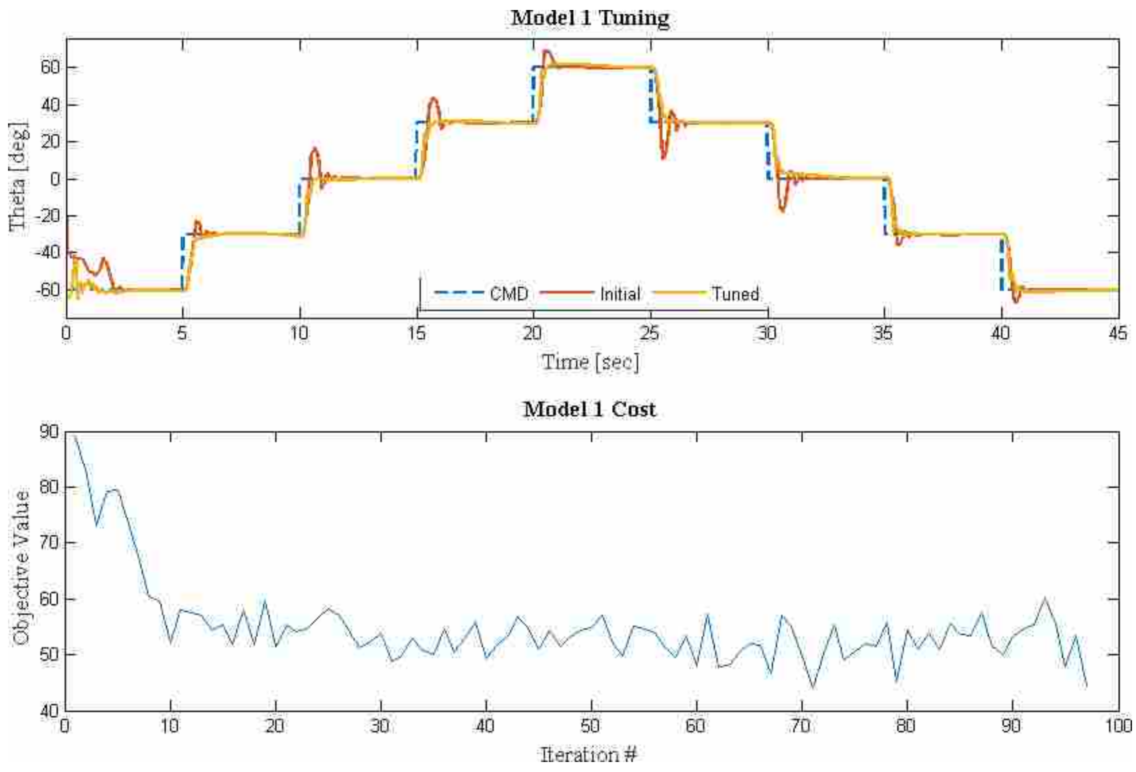

Figure 6.5: Parameter set 1 performance improvement and iteration cost

Tuned controller performance of both sets of parameters were overlayed with the original hand tuned controller in Figure 6.7. When compared to Figure 6.4, a dramatic overall improvement in angle control performance was seen.

6.4 Conclusion

Despite the underlying dynamic models used by MPC containing dynamic parameters with significant disturbances, when comparing angle control in Figure 6.4 to 6.7 a dramatic performance increase can be seen with both parameter sets after automated tuning. This improvement in performance demonstrated not only the significant effect of MPC weights on control performance, but also the robustness of MPC as a control scheme in the presence of model disturbances. The automated tuning process described in Section 6.2 was intended to be general and could potentially be used with any dynamic system utilizing MPC. The results of the trial within this chapter also suggest that automated tuning process could potentially be used to mitigate the gradual control

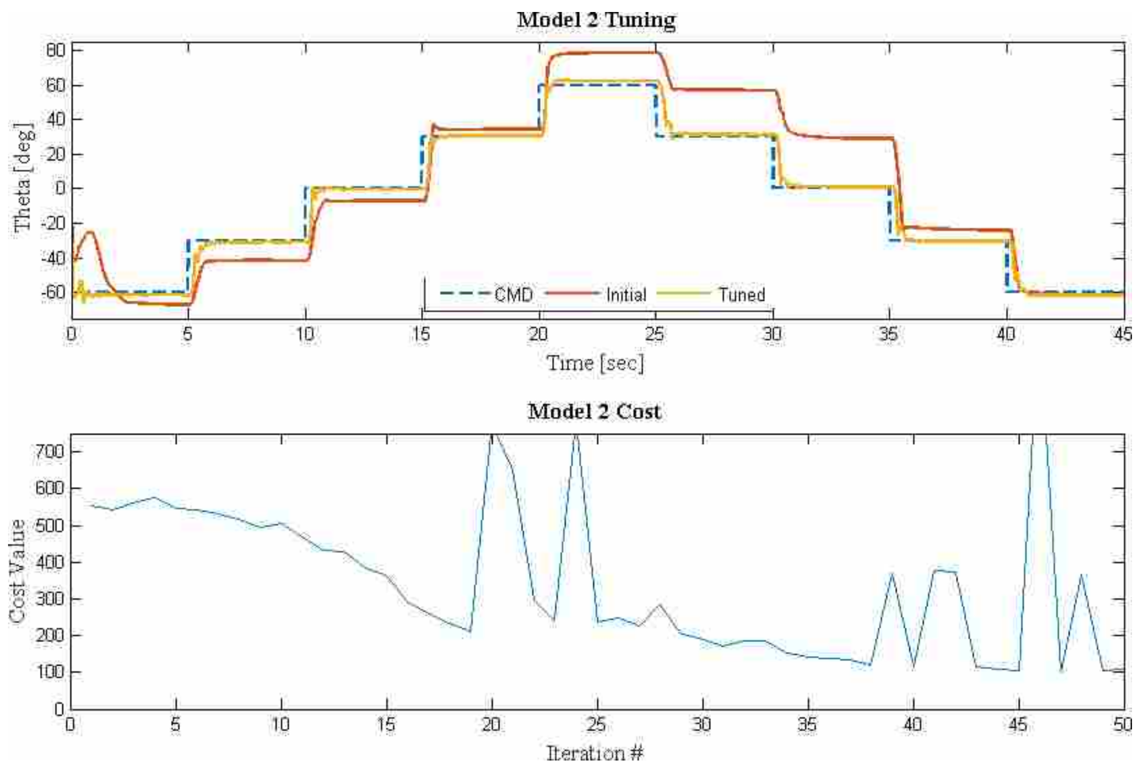

Figure 6.6: Parameter set 2 performance improvement and iteration cost

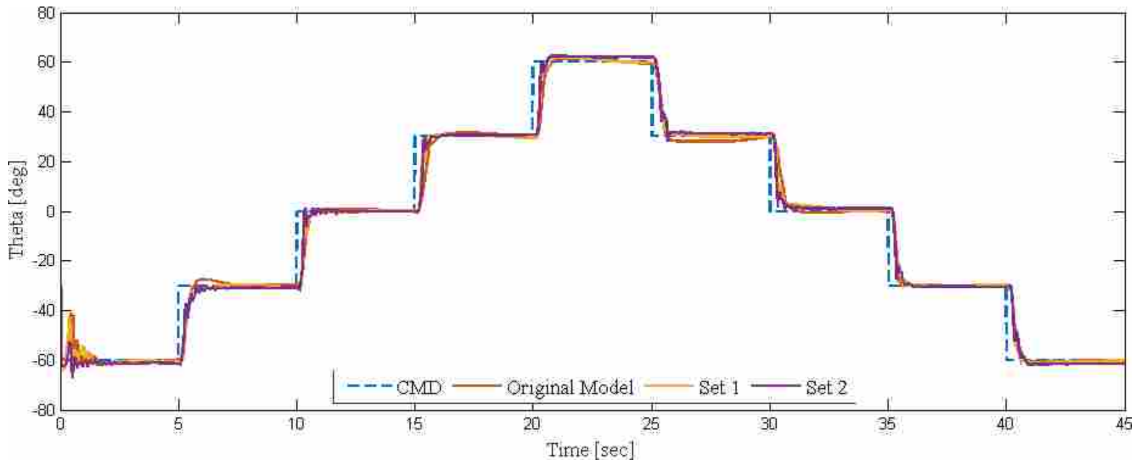

Figure 6.7: Comparison of tuned system performance

performance degradation present in the inflatable platforms used within this work, seen in Section 4.2.3 where unknown changes in system dynamics resulted in poor control performance.

CHAPTER 7. CONCLUSION

This thesis derived and compared multiple dynamic models for describing torque in soft antagonistic actuators. These actuators have most recently been seen in lightweight inflatable robotic platforms, where natural compliance was a primary consideration in model development. This inherent compliance and unknown relation from actuator fluid pressure to torque made dynamic descriptions and control of these platforms very limited. Using the descriptions for torque described within this thesis, several dynamic models were created, a model predictive controller was developed, and the controller performance tested on two fully inflatable, underdamped, and antagonistic, soft robotic platforms. An automated tool for tuning of model predictive controllers based on common performance measures was also developed and tested on one of these inflatable platforms. This thesis shows that the models and controller developed within have improved upon prior control methodologies used on these platforms in the past.

7.1 Future Work

Throughout this thesis, multiple single-DoF models were developed using the derived torque relations. These models did scale adequately well to multi-DoF, it is believed performance could be significantly improved if both gravitational and joint interference effects were considered. While the multi-DoF DNN model did consider the states of both joints when predicting future joint angle states, it was believed that one joint may have received unequal weight in the objective function and considered it more important to correctly track one angle over the other. Future work could weight this objective function differently or derive an individual net for each joint to improve model predictive capabilities. This could potentially be achieved by a differently designed network or weighted optimization objective function. Performance of the neural network model prediction could also be improved by extracting the output gradients directly, instead of using central finite differencing.

Control performance could potentially be improved by more accurate dynamic models, longer predictive horizons, or improved tuning. Computational limitations prevented the model predictive control solver from solving faster than 300 Hz while predicting a 20 time step horizon. Improved solvers or computational capabilities would allow for higher solving rate for further predictions. Higher rate control would allow for consideration of higher frequency dynamics, whereas a further predictive horizon would allow for better consideration of slower dynamics.

The tuning process of these controllers could also be improved through usage of a more apt optimization method or true multi-objective optimization. Using a method such as surrogate model optimization could result in fewer required iterations to achieve a reasonable minimum. Whereas developing a multi-objective Pareto front describing a few select performance characteristics would allow the end user to choose between a variety of optimal controller weights. This would allow the user to make trade off decisions; for example, choose between a controller that had lower rise time but low percent overshoot.

The DNN model could also potentially be fully expanded into a suboptimal model predictive controller itself. If the DNN model were instead used to predict multiple time steps into the future, the outputs of a series of predetermined input trajectories could all be calculated simultaneously. The input trajectory which predicted an output trajectory with the lowest calculated cost would be selected. This setup, while computationally expensive, has already been shown to work on a variety of systems.

Within this work, mutli-DoF inflatable joint angles were estimated through motion capture camera system determining link pose and known kinematic relationships. Future work could utilize these same formulations, but with joint poses acquired through integrated IMUs instead. This change would allow these platforms to operate outside the laboratory and in the environments described within the initial motivations: human environments.

7.2 Contributions

This thesis first describes the derivation of three new models for describing torque in an antagonistic soft fluidic actuator in Chapter 3. These torque descriptions were formed into separate state space models and their predictive capabilities compared in section 3.5. For one of the models described in section 3.4, a predictive deep neural network was constructed and adapted to produce usable dynamic parameters for a linear state space model. This chapter demonstrated that state prediction of inflatable robotic platforms with soft fluidic actuators was both reasonable and accurate.

In Chapter 4 a single-DoF model predictive controller was derived in section 4.1 which utilized the three models developed in Chapter 3. This controller was adapted to each of the different models and a tunable anti-windup integrator was developed in Section 4.2.2. Angle position trajectory tracking, as described in both [23] and [24], was used to compare controller tracking performance to prior state-of-the-art on the single-DoF Grub platform. General controller angle tracking performance was significantly improved over prior research with all controllers. The effects of controller performance change due to slight changes in hardware over time were described and quantified in section 4.2.3. The developed controller utilized a weighted value P_T within the cost function in equation 4.1 to raise pressure values above 0. The effects of changing this value on joint stiffness were described in 4.3. This chapter concluded that the developed model predictive controller could be used to track angular position accurately while also varying effective joint stiffness using the models developed in Chapter 3.

In chapter 5 the model predictive controller was expanded to control the multi-DoF King Louie platform. This expansion was accomplished by treating each joint as a separate, uncoupled 1-DoF joint. A larger DNN was developed in 5.2.3 to predict angle states of two joints simultaneously. The angle tracking performance of each controller was compared in section 5.3. This chapter highlighted the difficulties of moving these controllers to multiple degrees-of-freedom while also demonstrating that they were able to track an angle trajectory.

Finally in Chapter 6 an automated process for tuning of model predictive controller weights using common closed-loop controls performance criteria was developed and demonstrated. Two sets of dynamic parameters with significant disturbances were generated for the model described in section 3.1 and untuned controller performance was captured and compared. The automated tuner was applied to controllers using models with disturbances and their final tuned performance was compared to the hand tuned model in section 6.3. This chapter demonstrated that the weights used by the model predictive controller cost function could be tuned automatically using automated discrete performance measurements optimization. This chapter also highlighted the robustness of model predictive control in the presence of model inaccuracies.

REFERENCES

[1] Sanan, S., Moidel, J., and Atkeson, C., 2009. "Robots with inflatable links." In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 4331–4336. 2

[2] Sanan, S., Ornstein, M. H., and Atkeson, C. G., 2011. "Physical human interaction for an inflatable manipulator." In *Engineering in Medicine and Biology Society, EMBC, 2011 Annual International Conference of the IEEE*, IEEE, pp. 7401–7404. 2

[3] Bicchi, A., and Tonietti, G., 2004. "Fast and "soft-arm" tactics [robot arm design]." *Robotics Automation Magazine, IEEE,* **11**(2), June, pp. 22–33. 2

[4] Sanan, S., Lynn, P. S., and Griffith, S. T., 2014. "Pneumatic torsional actuators for inflatable robots." *Journal of Mechanisms and Robotics,* **6**(3), p. 031003. 2

[5] Carneiro, J. F., and de Almeida, F. G., 2006. "Reduced-order thermodynamic models for servo-pneumatic actuator chambers." *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering,* **220**(4), pp. 301–314. 2

[6] Shen, X., and Goldfarb, M., 2007. "Simultaneous force and stiffness control of a pneumatic actuator." *Journal of Dynamic Systems, Measurement, and Control,* **129**(4), pp. 425–434. 2

[7] Bicchi, A., Rizzini, S. L., and Tonietti, G., 2001. "Compliant design for intrinsic safety: General issues and preliminary design." In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, Vol. 4, IEEE, pp. 1864–1869. 2

[8] Tonietti, G., and Bicchi, A., 2002. "Adaptive simultaneous position and stiffness control for a soft robot arm." In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, Vol. 2, IEEE, pp. 1992–1997. 2

[9] Ivlev, O., 2009. "Soft fluidic actuators of rotary type for safe physical human-machine interaction." *2009 IEEE International Conference on Rehabilitation Robotics, ICORR 2009,* **28359**, pp. 1–5. 2

[10] Gaiser, I., Wiegand, R., Ivlev, O., Andres, a., Breitwieser, H., Schulz, S., and Bretthauer, G., 2014. "Compliant Robotics and Automation with Flexible Fluidic Actuators and Inflatable Structures." *Smart Actuation and Sensing SystemsRecent Advances and Future Challenges*, pp. 567–608. 2

[11] Qin, S. J., and Badgwell, T. A., 2003. "A survey of industrial model predictive control technology." *Control Engineering Practice,* **11**(7), July, pp. 733–764. 9

[12] Jain, A., Killpack, M. D., Edsinger, A., and Kemp, C. C., 2013. "Reaching in clutter with whole-arm tactile sensing." *The International Journal of Robotics Research.* 9

[13] Killpack, M. D., and Kemp, C. C., 2013. "Fast reaching in clutter while regulating forces using model predictive control." In *Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on*, IEEE. 9

[14] Killpack, M. D., Kapusta, A., and Kemp, C. C., 2015. "Model predictive control for fast reaching in clutter." *Autonomous Robots*, pp. 1–24. 9

[15] Erez, T., Tassa, Y., and Todorov, E., 2012. "Infinite-horizon model predictive control for periodic tasks with contacts." *Robotics: Science and Systems VII*, p. 73. 9

[16] Rupert, L., Hyatt, P., and Killpack, M. D., 2015. "Comparing model predictive control and input shaping for improved response of low-impedance robots." In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, IEEE, pp. 256–263. 9

[17] Shim, D. H., Kim, H. J., and Sastry, S., 2003. "Decentralized nonlinear model predictive control of multiple flying robots." In *Proceedings. 42nd IEEE Conference on Decision and Control*, Vol. 4, IEEE, pp. 3621–3626. 9

[18] Leung, C., Huang, S., Kwok, N., and Dissanayake, G., 2006. "Planning under uncertainty using model predictive control for information gathering." *Robotics and Autonomous Systems,* **54**(11), July, pp. 898–910. 9

[19] Annamalai, A. S. K., Sutton, R., Yang, C., Culverhouse, P., and Sharma, S., 2014. "Robust adaptive control of an uninhabited surface vehicle." *Journal of Intelligent & Robotic Systems*, pp. 1–20. 9

[20] Wang, Y., and Boyd, S., 2010. "Fast model predictive control using online optimization." *IEEE Transactions on Control Systems Technology,* **18**(2), March, pp. 267–278. 9

[21] Domahidi, A., Zgraggen, A. U., Zeilinger, M. N., Morari, M., and Jones, C. N., 2012. "Efficient interior point methods for multistage problems arising in receding horizon control." In *Proceedings of the 51st IEEE Conference on Decision and Control*, no. EPFL-CONF-181938. 9

[22] Mattingley, J., and Boyd, S., 2012. "Cvxgen: a code generator for embedded convex optimization." *Optimization and Engineering,* **13**(1), pp. 1–27. 10, 34

[23] Best, C. M., Wilson, J. P., and Killpack, M. D., 2015. "Control of a pneumatically actuated, fully inflatable, fabric-based humanoid robot." In *Humanoids, 2015 IEEE-RAS International Conference on*, IEEE. 14, 32, 64

[24] Gillespie, M. T., Best, C. M., and Killpack, M. D., 2016. "Simultaneous position and stiffness control for an inflatable soft robot." In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, IEEE. 14, 58, 64

[25] Best, C., Gillespie, M., Hyatt, P., Rupert, L., Sherrod, V., and Killpack, M., 2016. "Model predictive control for pneumatically actuated soft robots." *IEEE Robotics & Automation Magazine*. 14, 21, 50

[26] Best, C. M., Wilson, J. P., and Killpack, M. D., 2015 (accepted). “Control of a pneumatically actuated, fully inflatable, fabric-based humanoid robot.” In *Humanoids, 2015 IEEE-RAS International Conference on*, IEEE. 14, 20, 35

[27] Fabri, S., and Kadirkamanathan, V., 1996. “Dynamic structure neural networks for stable adaptive control of nonlinear systems.” *IEEE Transactions on Neural Networks,* **7**(5), pp. 1151 –1167. 24

[28] Meng, M., 1999. “A neural network approach to real-time motion planning and control of robot manipulators.” *IEEE SMC’99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028),* **4**, pp. 674–679. 24

[29] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X., 2015. “TensorFlow: Large-scale machine learning on heterogeneous systems.” Software available from tensorflow.org. 24

[30] Glorot, X., and Bengio, Y., 2010. “Understanding the difficulty of training deep feedforward neural networks.” *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS),* **9**, pp. 249–256. 25

[31] Kingma, D., and Ba, J., 2014. “Adam: A Method for Stochastic Optimization.” *International Conference on Learning Representations*, pp. 1–13. 27

[32] Tetko, I. V., Livingstone, D. J., and Luik, A. I., 1995. “Neural-Network Studies .1. Comparison of Overfitting and Overtraining.” *Journal of Chemical Information and Computer Sciences,* **35**(5), pp. 826–833. 27

[33] Anderson, B. W., 2001. *The Analysis and Design of Pneumatic Systems*. Krieger Publishing Company, 3. 42

[34] Olesen, D. H., 2012. “Tuning methods for model predictive controllers.” Master’s thesis, Technical University of Denmark, DTU Informatics, E-mail: reception@imm.dtu.dk, Asmussens Alle, Building 305, DK-2800 Kgs. Lyngby, Denmark Supervised by Associate Professor John B. Jørgensen, jbj@imm.dtu.dk, DTU Informatics. 53

[35] Garriga, J. L., and Soroush, M., 2010. “Model predictive control tuning methods: A review.” *Industrial & Engineering Chemistry Research,* **49**(8), pp. 3505–3515. 53, 57

[36] Levine, W., 2010. *The Control Handbook, Second Edition: Control System Applications, Second Edition*. Electrical Engineering Handbook. CRC Press. 53, 54