Constrained Nonlinear Heuristic-Based MPC for Control of Robotic Systems with Uncertainty

Tyler James Quackenbush

Brigham Young University

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

Part of the Engineering Commons

BYU ScholarsArchive Citation

Quackenbush, Tyler James, "Constrained Nonlinear Heuristic-Based MPC for Control of Robotic Systems with Uncertainty" (2021). Theses and Dissertations. 9288.
https://scholarsarchive.byu.edu/etd/9288

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

of Robotic Systems with Uncertainty

Tyler James Quackenbush

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

Master of Science

Marc Killpack, Chair
John Salmon
Mark Colton

Department of Mechanical Engineering
Brigham Young University

Copyright © 2021 Tyler James Quackenbush
All Rights Reserved
ABSTRACT

Constrained Nonlinear Heuristic-Based MPC for Control of Robotic Systems with Uncertainty

Tyler James Quackenbush
Department of Mechanical Engineering, BYU
Master of Science

This thesis focuses on the development and extension of nonlinear evolutionary model predictive control (NEMPC), a control algorithm previously developed by Phil Hyatt of the BYU RaD Lab. While this controller and its variants are applicable to any high degree-of-freedom (DoF) robotic system, particular emphasis is given in this thesis to control of a soft robot continuum joint. First, speed improvements are presented for NEMPC. Second, a Python package is presented as a companion to NEMPC, as a method of establishing a common interface for dynamic simulators and approximating each system by a deep neural network (DNN). Third, a method of training a DNN approximation of a hardware system that is generalize-able to more complex hardware systems is presented. This method is shown to reduce median tracking error on a soft robot hardware platform by 88%. Finally, particle swarm model predictive control (PSOMPC), a variant of NEMPC, is presented and modified to model and account for uncertainty in a dynamic system. Control performance of NEMPC and PSOMPC are presented for a set of control trials on simulated systems with uncertainty in parameters, states, and inputs, as well as on a soft robot hardware platform. PSOMPC is shown to have an increased robustness to system uncertainty, reducing expected collisions by 71% for a three-link robot arm with parameter uncertainty, input disturbances, and state measurement error.

Keywords: model predictive control, soft robot, uncertainty, deep neural net
ACKNOWLEDGMENTS

I thank my advisor and my committee for their time and help along the process of producing this thesis. I also thank my lab-mates for spending more of their time than was probably necessary staring at a whiteboard with me trying to debug these algorithms. I may have written the research down, but many of the key ideas and algorithms were the product of the time spent in collaboration with members of the RaD Lab. Most importantly, I thank my incredible wife Heather and our two boys for their patience and love over this process, and the many times when Heather listened to my frustrations about the research, then proceeded to solve the issues I was having without ever needing to see the code. All of these people have been key players in this thesis, and I can claim no credit without their support and guidance.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TITLE PAGE</td>
<td>i</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>ii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>iii</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td><strong>Chapter 1  Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Background</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Contributions and Thesis Outline</td>
<td>4</td>
</tr>
<tr>
<td><strong>Chapter 2  Nonlinear Evolutionary Model Predictive Control</strong></td>
<td>6</td>
</tr>
<tr>
<td>2.1 NEMPC speedup</td>
<td>8</td>
</tr>
<tr>
<td>2.2 RaD Models Package and Neural Net Approximation Module</td>
<td>8</td>
</tr>
<tr>
<td>2.3 Better DNN Hardware Approximations</td>
<td>11</td>
</tr>
<tr>
<td>2.3.1 Related Work</td>
<td>11</td>
</tr>
<tr>
<td>2.3.2 Soft Robot Model Definition</td>
<td>13</td>
</tr>
<tr>
<td>2.3.3 Surrogate and Error DNN Training</td>
<td>15</td>
</tr>
<tr>
<td>2.4 NEMPC Control Performance</td>
<td>17</td>
</tr>
<tr>
<td>2.4.1 Nonlinear Evolutionary Model Predictive Control</td>
<td>17</td>
</tr>
<tr>
<td>2.4.2 Hardware Experiments</td>
<td>21</td>
</tr>
<tr>
<td>2.4.3 Hardware Results</td>
<td>22</td>
</tr>
<tr>
<td>2.5 Conclusion</td>
<td>25</td>
</tr>
<tr>
<td><strong>Chapter 3  Optimal Control for Nonlinear Robotic Systems in the Presence of Uncertainty</strong></td>
<td>27</td>
</tr>
<tr>
<td>3.1 Motivation</td>
<td>27</td>
</tr>
<tr>
<td>3.2 Introduction</td>
<td>27</td>
</tr>
<tr>
<td>3.3 Related Work</td>
<td>29</td>
</tr>
<tr>
<td>3.4 Nonlinear Heuristic Model Predictive Control Formulations</td>
<td>31</td>
</tr>
<tr>
<td>3.4.1 NEMPC</td>
<td>34</td>
</tr>
<tr>
<td>3.4.2 PSOMPC</td>
<td>35</td>
</tr>
<tr>
<td>3.5 Simulation Experiments</td>
<td>39</td>
</tr>
<tr>
<td>3.5.1 Nonlinear Mass-Spring-Damper</td>
<td>40</td>
</tr>
<tr>
<td>3.5.2 Three-link Robot Arm</td>
<td>45</td>
</tr>
<tr>
<td>3.6 Soft Robot Experiments</td>
<td>52</td>
</tr>
<tr>
<td>3.6.1 System Definition</td>
<td>52</td>
</tr>
</tbody>
</table>
3.6.2 Simulation Results and Discussion ........................................... 58
3.6.3 Hardware Results and Discussion ........................................... 59
3.7 Conclusion ............................................................................. 61

Chapter 4 Conclusions and Future Work ................................................. 66
4.1 Future Work ............................................................................. 66
  4.1.1 RaD Models ......................................................................... 66
  4.1.2 DNN Modeling .................................................................... 67
  4.1.3 Dynamic Particle Swarm MPC ............................................. 67
4.2 Conclusion ............................................................................. 68

REFERENCES .................................................................................. 69

Appendix A Details of Parallelization to Improve Execution Time for NEMPC .......... 74
LIST OF TABLES

2.1 RaD Models Interfaces ............................................ 10
2.2 Comparison of control performance of NEMPC with error compensation versus NEMPC without error compensation. ........................................ 23

3.1 Hyperparameters for NEMPC Algorithm .................................. 36
3.2 Hyperparameters for PSOMPC Algorithm .................................. 38
3.3 Uncertainty Scenarios for Simulated Experiments ......................... 39
3.4 Mass Spring Damper Parameters ....................................... 43
3.5 NEMPC Hyperparameters for Mass-Spring-Damper Experiments .......... 43
3.6 PSOMPC Hyperparameters for Mass-Spring-Damper Experiments ........... 44
3.7 Three-Link Robot Arm Parameters ..................................... 51
3.8 NEMPC Hyperparameters for Three-Link Experiments .................... 51
3.9 PSOMPC Hyperparameters for Three-Link Experiments .................. 52
3.10 NEMPC Hyperparameters for Soft Robot Continuum Joint Experiments .... 56
3.11 PSOMPC Hyperparameters for Soft Robot Continuum Joint Experiments ... 57
LIST OF FIGURES

1.1 The general structure of optimal model-based control. In this diagram, it is assumed that $x_t$ is known, not measured. ................................................................. 2

1.2 Representation of parallelized model predictive control. Each of the three sets of predicted inputs corresponds to a predicted output. Figure adapted with permission from [1]. 3

1.3 Representation of parallelized model predictive control with a knot point approximation for each predicted input. As in Figure 1.2, each set of predicted inputs corresponds to a predicted output. The predicted inputs are interpolated for simulation as shown in Figure 1.4. Figure adapted with permission from [1]. ........................................ 4

1.4 The interpolation of a knot point input series. Each step in time represents a $\Delta t$ for the simulation, and one interpolated input is applied at each discrete time step. .......... 4

2.1 Evaluation time for NEMPC over the course of several revisions. ‘Original Code’ refers to the code developed Hyatt et al. [2]. ‘Revision 1’ included parallelization of the mutation routine, while ‘Revision 2’ included parallelization of the mating and mutation routines. ‘Current Code’ incorporates the parallelization of the CPU to GPU data transfer necessary for the DNN evaluation. ........................................ 9

2.2 Photograph of soft robotic continuum joint used for this work. ........................................ 14

2.3 Control diagram for running NEMPC in conjunction with the learned error model. $u^*$ indicates the optimal input chosen by the controller. This input is sent to the embedded pressure controller and we measure pressures $p$ and positions $q$ directly, while estimating $\dot{q}$. ................................................................. 16

2.4 Comparison of pressure dynamics between the four different systems used. The dashed line indicates the commanded pressure, while each of the solid lines is the pressure response resulting from the commanded pressure input. Note that the states from the analytical model and the surrogate DNN match well and that when using the combined DNN, the simulation closely resembles the hardware data. ........................................ 17

2.5 Comparison of velocity dynamics between the four different models used. These velocities are the response resulting from the commanded pressure inputs shown in Figure 2.4. Note that the surrogate DNN tracks the analytical model well while the combined DNN tracks the hardware data. ........................................ 18

2.6 Comparison of joint angle dynamics between the four different models used. These angles are the response resulting from the commanded pressure inputs shown in Figure 2.4 ................................................................. 19

2.7 Diagram of the experimental setup for the hardware experiments. Also illustrated here is the inherent plasticity of the robot, resulting in a variable offset in $\theta$ and $\phi$. Over time, the plastic in the pressure chambers deforms and causes the robot to have an equilibrium configuration that is not vertical. ........................................ 21

2.8 Comparison of tracking performance on the physical soft robot continuum joint while using the two categories of DNN model approximation. Note that the control performance of NEMPC while using the combined DNN contains much less steady-state tracking error than the control performance of NEMPC while using the surrogate DNN. 22
2.9 A histogram of the normalized frequency of $\theta$ and $\phi$ tracking error in the hardware experiments. Note that the data gathered while using the surrogate DNN for control has $\theta$ error and $\phi$ error that is biased in both directions away from zero. This is a result of the surrogate DNN’s lack of information regarding the offsets in $\theta$ and $\phi$ at equilibrium. Also note the difference in y axis scaling for both histograms.  

2.10 Comparison of tracking performance on sine (left column) and ramp (right column) test signals using the $N_{\text{sim}} + N_{\text{err}}$ DNN configuration for control. Note that although both DNNs were trained solely on step inputs, the models are able to generalize well to other types of signals.  

3.1 The constrained nonlinear mass-spring-damper system, with a wall (barrier) located at $x_{\text{wall}} = 1.0$. The system is pictured at rest, with $x = 0$.  

3.2 Friction force as a function of $\dot{x}$ for a mass-spring-damper with $\mu = 0.1$, $m = 3.0$.  

3.3 Performance of PSOMPC and NEMPC on a constrained mass spring damper model under the five uncertainty scenarios. Data from 500 two-second rollouts are presented for NEMPC and PSOMPC in each uncertainty scenario, for a total of 5000 rollouts.  

3.4 Mean and standard deviation of linear mass spring damper performance with no uncertainty in the system. Data from 500 two-second rollouts are presented for NEMPC and PSOMPC.  

3.5 Mean and standard deviation of PSOMPC and NEMPC control of a non-linear mass spring damper with varying parameters. Data from 500 two-second rollouts are presented for NEMPC and PSOMPC.  

3.6 Mean and standard deviation of PSOMPC and NEMPC control of a linear mass spring damper with wandering noise on the system input. Data from 500 two-second rollouts are presented for NEMPC and PSOMPC.  

3.7 Mean and standard deviation of PSOMPC and NEMPC control of a linear mass spring damper with noise applied to state (position and velocity) measurements. Data from 500 two-second rollouts are presented for NEMPC and PSOMPC.  

3.8 Mean and standard deviation of PSOMPC and NEMPC control of a non-linear mass spring damper with all forms of uncertainty combined. Data from 500 two-second rollouts are presented for NEMPC and PSOMPC.  

3.9 The three-link robot arm system, with an obstacle located at $x_{\text{wall}} < -0.1$ & $y_{\text{wall}} > 1.25$. The system is pictured at rest, with $q = [\pi, 0, 0]^T$. The pivot for link 0 is at the origin.  

3.10 Performance of PSOMPC and NEMPC on a constrained three link robot arm model under various conditions of system uncertainty. Data from 500 rollouts are presented for NEMPC and PSOMPC in each uncertainty scenario, representing 5000 rollouts in total.  

3.11 Angle data for the three link robot arm experiments with no uncertainty in the system. This figure contains data from 500 rollouts for each algorithm.  

3.12 Path data for PSOMPC and NEMPC control of a three link robot arm with varying parameters. This figure contains data from 500 rollouts for each algorithm.  

3.13 Path data for PSOMPC and NEMPC control of a three link robot arm with wandering noise on the system input. This figure contains data from 500 rollouts for each algorithm.  

viii
3.14 Path data for PSOMPC and NEMPC control of a three link robot arm with noise applied to state (position and velocity) measurements. This figure contains data from 500 rollouts for each algorithm. ................................................................. 57
3.15 Path data for PSOMPC and NEMPC control of a three link robot arm with all forms of uncertainty combined. This figure contains data from 500 rollouts for each algorithm. 58
3.16 Performance of PSOMPC on a constrained three link robot arm model under various conditions of system uncertainty. For each value of $\sigma_{\text{noise}}$ shown in the legend, 500 two-second experiments were conducted in the same manner as those in Figure 3.10, for a total of 20,000 rollouts. ................................................................. 59
3.17 Performance of PSOMPC on a constrained three link robot arm model under various conditions of system uncertainty. For each value of $n_{\text{swarms}}$ shown in the legend, 500 two-second experiments were conducted in the same manner as those in Figure 3.10. Note that increasing the number of swarms does not increase the size of PSOMPC’s population, it merely splits the total population into more individual groups. For each value shown in the legend, 500 two-second experiments were conducted in the same manner as those in Figure 3.10, for a total of 20,000 rollouts. ................................. 60
3.18 An image of the experimental setup of the soft robot continuum joint with its spherical tip and an obstacle in its workspace. ................................................................. 61
3.19 A diagram of the experimental setup for the soft robot continuum joint. A hanging obstacle lies within the robot’s workspace and is reachable with its spherical tip. Contact between the obstacle and spherical tip is considered a collision ................................. 62
3.20 Performance of PSOMPC and NEMPC for 50 rollouts of an obstacle approach task on a simulated soft robot continuum joint. ................................................................. 63
3.21 $\phi$ and $\theta$ history from the 50 rollouts of the simulated soft robot continuum joint experiment. ................................................................. 64
3.22 Performance of PSOMPC and NEMPC for an obstacle approach task on the soft robot continuum joint. Note that the ‘barrier’ as recorded in software is slightly larger than the obstacle in hardware, therefore a ‘collision’ in software does not always coincide with a collision in hardware. This figure reports physical collisions, not software collisions. Data from 50 rollouts for each algorithm is presented here. ......................... 64
3.23 A histogram of the maximum penetration into the obstacle region as defined in software. Both NEMPC and PSOMPC crossed the barrier boundary as defined in software multiple times, but NEMPC penetrates the obstacle farther and with a greater frequency. 65
3.24 $\phi$ and $\theta$ history from the 50 rollouts of the hardware experiment. The software ‘barrier’ is larger than the obstacle in hardware, therefore crossing the red dashed line is not necessarily indicative of a collision in hardware. Data from 50 rollouts for each algorithm is presented here. ................................. 65

A.1 An example of Numpy’s broadcasting pitfalls. ................................................................. 74
A.2 Setting up broadcasting with mismatched dimensions. ................................................................. 75
A.3 When broadcasting, data is copied to fill the missing dimension. ................................................................. 75
A.4 An example of two arrays of similar shapes, but one with a singleton dimension. ................................................................. 76
A.5 When broadcasting, data is copied to expand the singleton dimension. ................................................................. 76
A.6 The shapes of the data structures used in NHMPC. ................................................................. 77
CHAPTER 1. INTRODUCTION

As robots become more general-purpose and extend their utility outside of traditional automation tasks, they require greater complexity of motion. This complexity of motion results in a large number of degrees-of-freedom (DoF), and makes the system difficult to control. Optimal model-based control of high-DoF systems is challenging due to the complexity of modeling these systems, but becomes increasingly difficult if these systems have inherent uncertainty in their parameters, uncertainty in the measurement of their states, or input disturbances. These robots also may experience uncertainty or constraints in their environment, such as when locomoting across uneven terrain, handling variable payload situations, and operating in close proximity to humans. Such situations make optimal model-based control of robotic systems difficult, but also require the predictive nature of an optimal controller in order to allow these systems to operate effectively.

In order to leverage the utility of high-DoF robots in these uncertain scenarios, we developed model-based optimal controllers that can both predict real hardware behavior and account for a system’s uncertainty in the presence of constraints. For this thesis, we utilize soft robots as a test bed platform for high-DoF systems with uncertainty. Soft robots have highly nonlinear dynamics, uncertainty in their parameters, and deform under disturbances, making them an ideal test-bed for these scenarios. We control these robots using nonlinear heuristic formulations of model predictive control (MPC), and develop methods for accounting for unmodeled uncertainty in the systems.

The primary objective of this thesis is to expand on the nonlinear evolutionary model predictive control (NEMPC) algorithm developed by Hyatt et al. [2], a variant of model predictive control (MPC) that uses a genetic algorithm (GA) to calculate an optimal series of inputs to a system that drive the system to a set of goal states. This thesis presents a series of improvements to the NEMPC algorithm to increase speed and ease of use, as well as introduces particle swarm model predictive control (PSOMPC), a variant of NEMPC which is designed to have a greater robustness to uncertainty than previous methods.
1.1 Background

This thesis explores several variants of optimal model-based control, each derived from model predictive control (MPC). The general structure for using an optimal model-based controller is presented in Figure 1.1, where the controller retains an internal model of the plant to be controlled. The controller uses that internal model to forward predict over a time horizon, calculating an optimal input $u^*$ to apply to the plant. This process repeats and a new $u^*_t$ is calculated at each time step.

![Diagram of optimal model-based control](image)

Figure 1.1: The general structure of optimal model-based control. In this diagram, it is assumed that $x_t$ is known, not measured.

For all MPC algorithms presented in this thesis, the optimization conducted to calculate $u^*_t$ takes the form of a parallelized heuristic optimization. In such an optimization, the controller maintains a ‘population’ or ‘swarm’ of predicted inputs over some time horizon, as seen in Figure 1.2. Each blue set of predicted inputs is simulated on the controller’s internal model over the time horizon, resulting in a set of predicted outputs (green lines in Figure 1.2). For this thesis, the controller’s internal model of the plant is always a deep neural net (DNN) trained to approximate the plant dynamics. By so doing, the model is able to be evaluated for all ‘population’ or ‘swarm’ members in parallel on a graphics processing unit (GPU), decreasing computation time by orders of magnitude (see Chapter 2 for more details).

Using the predicted inputs, predicted outputs, and a goal state, a cost function supplied to the optimization assigns a cost to each predicted input series in the ‘swarm’ or ‘population’. The
set of predicted inputs with the lowest cost can be taken as an approximately optimal trajectory, with the first input of that trajectory selected as \( u_t^* \) for that time step.

All controllers presented in this thesis also utilize a ‘knot-point approximation’ of the predicted inputs, as shown in Figure 1.3. This concept was introduced in [1], and consists of approximating each set of predicted inputs as a series of knot points, making each predicted input series parameterized by fewer data points. When simulating on the controller’s internal model, each set of predicted inputs is interpolated as shown in Figure 1.4 when generating the predicted outputs of the system. By so doing, the dimensionality of the optimization is reduced, allowing the optimization to explore a smaller optimization space and find optima more quickly.

Between time \( t \) and \( t + 1 \), the controller uses its knowledge of the predicted inputs and their cost to generate a new ‘population’ or ‘swarm’ of predicted inputs, and the process repeats. The manner in which the controller obtains those new inputs is defined for each controller, and will be discussed in-depth in Chapters 2 and 3. The algorithm by which new ‘populations’ and ‘swarms’ are generated forms the core of any given control algorithm, and the methods for doing so are primary contributions of this thesis.
1.2 Contributions and Thesis Outline

The contributions of this thesis are as follows:

1. Development of particle swarm model predictive control (PSOMPC), a variant of NEMPC utilizing the particle swarm optimization routine and incorporating hyperparameters for the modeling of system uncertainty
2. An updated revision of NEMPC, with parallelized mating and mutation for a large decrease in computation time

3. A method for obtaining better DNN approximations of the dynamics of hardware systems

4. The RaD Models package, a collection of system models with a common interface for use with model-based controllers

5. The DNN Approximation module inside the RaD Models package, a module that abstracts away the need for expertise in understanding DNN architecture and allows for DNN training and initialization of RaD Models with a simple interface

Items 2-5 are discussed in Chapter 2, which first describes a high-level description of our heuristic-based algorithms which encompasses the methods presented in both Chapters 2 and 3. Items 2, 4, and 5 are necessary steps towards a journal paper detailing item 3, also presented in Chapter 2. This paper was a collaborative work with Curtis Johnson (BYU RaD Lab) and Taylor Sorensen (BYU Perception, Control, and Cognition Lab). Item 1 is presented in Chapter 3, which is a self-contained journal paper in preparation detailing the derivation of PSOMPC and an analysis of its control performance compared to NEMPC when operating under significant uncertainty.
CHAPTER 2. NONLINEAR EVOLUTIONARY MODEL PREDICTIVE CONTROL

This chapter focuses on improvements made to the nonlinear evolutionary model predictive control (NEMPC) algorithm. We first explain at a high level the nonlinear heuristic-based model predictive control paradigm which encompasses both of the controllers presented in this thesis. Then, in Sections 2.1 and 2.2 we describe improvements to NEMPC and an accompanying dynamic simulation package that enabled the work done in Sections 2.3 and 2.4. Section 2.3.2 presents our hardware platform, the analytical model used to generate training data, our deep neural network (DNN) training methods, and evaluation of each model’s accuracy. Section 2.4 explains the nonlinear evolutionary model predictive control (NEMPC) algorithm we employ and shows the results of our experiments and explores their implications. Section 2.5 discusses the importance of this work as well as current limitations and future directions for additional research.

Hyatt et al. originally developed NEMPC as a standalone controller, but to make the extension to other heuristic-based algorithms simpler (such as described in Chapter 3), we have adapted NEMPC to be a special case of a more general algorithm we choose to call nonlinear heuristic model predictive control (NHMPC). NHMPC serves as a general controller structure compatible with most swarm or population-based heuristic optimization algorithms, as will be demonstrated in Chapter 3. The NHMPC algorithm is presented in Algorithm 1, with the NEMPC generational update in Algorithm 2.

In Algorithms 1 and 2, $P$ represents the entire population of solutions in a normalized optimization space. Each member of $P$ represents $m$ inputs across a time horizon parameterized by $k$ knot points, as presented in [2]. $\mathcal{U}$ is the uniform distribution, and $J$ is the cost of a given input trajectory. $T$ represents the time horizon over which MPC will evaluate cost, and $\text{get}_u\_\text{from}_P$ interpolates the parameterized trajectories and maps the solution from optimization space ($p$) to system input space ($u$). $N$ represents the DNN approximation of the plant’s dynamics, which returns a $\Delta x$ to be added to $x_t$. $\text{get}_\text{new}_\text{generation}$ performs one step of the dynamic optimization
Algorithm 1 Nonlinear Heuristic Model Predictive Control Algorithm

1: for every simulation in parallel do
2: if Cold Start then
3: \( P = \mathcal{U}(p_{\min}, p_{\max}) \)
4: else if Warm Start then
5: \( P = \text{get}_{-}\text{new}_{-}\text{generation}(P, J) \)
6: end if
7: \( J = 0 \)
8: for \( t = 0 \) to \( T \) do
9: \( u_t = \text{get}_{-}\text{u}_{-}\text{from}_{-}P(P, t, T) \)
10: \( x_{t+1} = x_t + N(x_t, u_t) \)
11: \( J = J + \text{cost}_{-}\text{function}(x_t, u_t, t) \)
12: end for
13: end for
14: \( P^* = \text{get}_{-}\text{best}_{-}P(P) \)
15: \( u^* = \text{get}_{-}\text{u}_{-}\text{from}_{-}P(P^*, 0, T) \)
16: Apply \( u^* \) to the robot

Algorithm 2 Nonlinear Evolutionary Model Predictive Control Generation Update

1: \( P_{\text{parents}} = P \)'s with lowest \( J \)'s
2: for every child in parallel do
3: Randomly select two parents from \( P_{\text{parents}} \)
4: \( P_{\text{child}} = \text{crossover}(P_{\text{parent1}}, P_{\text{parent2}}) \)
5: if \( \mathcal{U}(0, 1) < p_{\text{mutate}} \) then
6: \( \sigma = \sigma_{\text{noise}} e \)
7: \( P_{\text{child}} = P_{\text{child}} + \mathcal{N}(\mu = 0, \sigma) \)
8: end if
9: end for
10: \( P_{\text{strangers}} = \mathcal{U}(p_{\min}, p_{\max}) \)
11: \( P = \text{concatenate}(P_{\text{parents}}, P_{\text{strangers}}, P_{\text{children}}) \)

and is defined in Algorithm 2. \( P^* \) represents the best solution currently located by the optimization, and is retrieved by \( \text{get}_{-}\text{best}_{-}P \), which returns the population member with the lowest cost.

In Algorithm 2, where we define the specific update method for a generation in NEMPC, the optimizer performs a genetic algorithm update to obtain the next generation of solutions. \( \text{crossover} \) constructs a \( P_{\text{child}} \) by taking individual real-valued genes from \( P_{\text{parent1}} \) or \( P_{\text{parent2}} \) with probability 0.5. \( p_{\text{mutate}} \in (0, 1) \) represents a probability of a given child being mutated. \( \sigma_{\text{noise}} \) is a hyperparameter that controls the magnitude of mutation noise applied to mutated children. \( \sigma \) is the noise applied at a given time step, and decays to zero as tracking error approaches zero. \( e \) is the tracking
error. \( \mathcal{N} \) is the normal distribution, and \textit{concatenate} concatenates sub-populations into one full population.

2.1 NEMPC speedup

NEMPC as developed by Hyatt et al. ran at a suitable rate, and enabled excellent control performance [2]. However, the code, which was implemented in Python using the Numpy scientific computing library, was not optimized for parallel processing of the optimization. As part of the restructuring of NEMPC, the code was vectorized - a process where unnecessary \textit{for} loops are replaced with matrix and block operations performed on large data structures. As a result, NEMPC’s mating and mutation phase dropped in necessary computation time by two orders of magnitude, as seen in Figure 2.1. With this reduction in solve time, NEMPC was able to control the soft robot continuum joint described later in this chapter at a rate of 100 Hz with a time horizon of 0.1 seconds. In addition, NEMPC execution time is now much less sensitive to population size, with a population increase only slightly increasing the controller execution time. For control of a three-link robot arm, a population size of 100 results in a solve time of 6.4 ms, while a population size of 1000 results in an 8.4 ms solve time. Further details of the process of this optimization can be found in Appendix A.

2.2 RaD Models Package and Neural Net Approximation Module

As part of the modifications to NEMPC, an accompanying software library for dynamic simulations was produced. These simulations were designed to be black-box compatible with the NEMPC algorithm (and the PSOMPC algorithm introduced in Chapter 3). This interface is designed to be extensible to other forms of model-based controllers we will develop in the future. The primary points of interface are described in Table 2.1.

Given the interfaces defined in Table 2.1, most model-based controllers should be capable of interfacing easily with the RaD Models package, with linear controllers accessing the \( \mathbf{A} \) and \( \mathbf{B} \) matrices and \( \mathbf{w} \) vector, and nonlinear controllers utilizing the \textit{calc_state_derivs} or \textit{forward_simulate_dt} methods to simulate model behavior.
Figure 2.1: Evaluation time for NEMPC over the course of several revisions. ‘Original Code’ refers to the code developed by Hyatt et al. [2]. ‘Revision 1’ included parallelization of the mutation routine, while ‘Revision 2’ included parallelization of the mating and mutation routines. ‘Current Code’ incorporates the parallelization of the CPU to GPU data transfer necessary for the DNN evaluation.

The existence of the `calc_state_derivs` and `forward_simulate_dt` methods also fit well with another prerequisite for NEMPC and PSOMPC: a deep neural net (DNN) approximation of the system’s dynamics. NEMPC and PSOMPC utilize a DNN to predict a system’s behavior with many possible trajectories, using the batch processing capabilities of DNNs to simulate multiple potential control inputs across long time horizons very rapidly. Utilizing the common interface of `calc_state_derivs` and `forward_simulate_dt`, the RaD Models package was expanded to incorporate a DNN approximation module that is capable of handling DNN training, loading, and evaluation without requiring user expertise.

To create a DNN approximation for a given RaD Model, a one-time script must be run for the RaD Model on a computer with a graphics processing unit (GPU). This four-line script will utilize the RadTrainer class to generate simulation data and use that data to train a DNN that approximates the system dynamics. This data is generated continuously, negating the effects of overfitting. The control inputs used to generate the data are square waves, sine waves, triangle...
waves, and gaussian noise, all across a wide range of frequencies, to negate the frequency sensitive effects reported in section 2.5. As the DNN is trained, parameters and DNN weights are saved continuously in the background, so training can be cancelled at any time and still leave the user with a valid set of DNN weights and parameters. For more complex systems, neural net hyperparameters can be adjusted through the RadTrainer and are saved with the DNN weights.

With a valid set of DNN weights and parameters, using the DNN is as simple as instantiating a RaD Model object with the flag `DNNApproximate = True`. The package will load all parameters and weights in the background and enable the user to interface with the model using the same interface as defined in Table 2.1, albeit requiring \( \mathbf{x} \) and \( \mathbf{u} \) to be Pytorch tensor objects loaded to the GPU.

This interface enables NEMPC to easily utilize pre-trained DNN approximations of systems, and generalizes well to training DNN approximations of models of hardware systems, but does nothing to improve the estimation errors present when approximating a hardware system by a
dynamic model. Sections 2.3, 2.4, and 2.5 contain excerpts from a journal paper created in collaboration with Curtis Johnson and Taylor Sorensen. We developed a DNN training method to better approximate real hardware dynamics, including learning unmodeled disturbances. Curtis Johnson and I collaborated on data gathering, processing, and control structure, while Taylor Sorensen made contributions in terms of DNN architecture and training.

2.3 Better DNN Hardware Approximations

In this section, the main contribution presented is a methodology for learning model discrepancies for use in NEMPC. This work was performed in collaboration with Curtis Johnson (BYU RaD Lab) and Taylor Sorensen (BYU Perception, Control, and Cognition Lab). We validate this approach in simulation and on a soft robot platform. This platform is an ideal test bed for our approach because the actual dynamics (both in terms of joint configuration and air pressure in the joint chambers over time) are intrinsically more uncertain than previously presented rigid robot systems and control methods discussed in Section 2.3.1. While we apply our approach to soft robotics to demonstrate its potential to learn both uncertain and unknown dynamics, the proposed method could generalize to any platform using a model predictive controller.

2.3.1 Related Work

The many desirable characteristics of soft robots present challenging problems when it comes to modeling and controlling them. Accurate physics-based (first-principles) models that are tractable for real-time model-based control are difficult to obtain because of uncertain material properties, hysteresis, nonlinear dynamics, and complicated pneumatic flow dynamics. Soft robot physics-based modeling efforts range from finite element (FEM) approaches as in [3] and [4] to Cosserat Rod models as in [5] or piecewise constant curvature (PCC) models as in [6] and [7]. Many of these methods have shown promise. However, the effort and expertise required to accurately model all of the aforementioned effects are formidable. Even if a perfectly accurate analytical model could be derived, it may be useless for real-time model-based control due to the high computational time required for evaluation, as will be shown in the experiments of Section 2.4.3. Additionally, even if the model is made tractable using appropriate simplifications, it would
likely still require significant effort in system identification to obtain acceptable closed loop control performance.

[8] and [9] both summarize the wide spectrum of strategies that have been proposed to overcome the aforementioned modeling challenges. Among these, data-driven modeling specifically addresses many difficulties of physics-based modeling for control. Generally, data-driven control algorithms are based on various forms of machine learning such as neural networks as in [10] and [11], Gaussian processes (GP) in [12], [13], [14], and [15], reinforcement learning (RL) as in [16], or sparse optimization (also known as SINDY) as in [17]. Notably, deep learning has proven to be a valuable tool for robot modeling and control and is explored thoroughly in [18] and [19]. Deep learning has more recently demonstrated the ability to approximate soft robot dynamic models accurately in [20, 21]. A major benefit of such approaches is that they are largely data-driven and as such, do not require an analytical model or specialized expertise. However, using these learned models in a real-time, model-based control formulation for soft robots (such as in [20, 21]) has been explored to a much lesser extent. Specifically, by using specialized hardware for accelerated computing, such as Graphics Processing Units (GPUs), data-driven models can be forward sampled in large batches and at high rates using a parallelized architecture. This enables their direct use to solve an optimal control problem using a nonlinear model predictive control strategy (see [22, 23]). This is the approach on which we build for this chapter.

On the other hand, an undesirable characteristic of data-driven modeling techniques is the need for large amounts of representative data, which are difficult to collect on hardware platforms where exploring the whole state space of the robot is infeasible or dangerous. Our approach in this chapter is to use a simplified, first-principles model to train a deep neural network (DNN) to represent general trends in state variables for the dynamics, and then add another deep neural network to compensate for additional error in the predicted states. To accomplish this, while also benefiting from the parallel computation available on a GPU, we first train a DNN to learn the first-principles model. Then we train a second DNN to learn the simulation-to-reality error gap. Because the first-principles DNN learns the general form of the dynamics from simulation, much less hardware training data is required. The hardware data only serves to make adjustments to capture unmodeled dynamics and does not necessarily need to be as representative or as plentiful as would be required if hardware data was exclusively used to train the neural network.
Our work towards compensating for modeling error with data-driven learning is similar to [24] where authors use deep learning to predict physics-based modeling error of water resources, [25] where they present an algorithm to learn a discrepancy model on an double inverted pendulum, and [7] where the authors augment a model-based disturbance observer with a learned correction factor on a soft robot. Most similar to our work is that of [26] where they augment a nonlinear model predictive controller with various forms of learned actions to compensate for model-plant mismatch on a rigid humanoid robot. Other works that include using neural networks as the backbone for predictive control are [27] and [28].

2.3.2 Soft Robot Model Definition

We start by providing an overview of our approach and how it fits with the methods and hardware presented in subsequent sections. Our overall approach to compensate for unknown modeling errors starts with training a deep neural network to act as a surrogate for the analytical model derived later in this section. This surrogate DNN is needed to exploit the parallelized architecture of modern GPUs, which in turn, affords higher control rates for our nonlinear MPC algorithm described in Section 2.4.1. Without approximating the analytical model via the surrogate DNN, the analytical model for the soft robot continuum joint is intractable for real-time control, and requires orders of magnitude more time for evaluation when compared to the DNN (see Table 2.2 in Section 2.4.3). Details related to the training of the surrogate DNN and error DNN are presented in Section 2.3.3.

Once the surrogate and the error DNN are trained we evaluate both in parallel, resulting in a combined forward prediction model (that we refer to as a combined DNN) which reflects the dynamics of the hardware platform more accurately. By improving the forward prediction capabilities of our model, we enable the controller to find more optimal input trajectories and thereby improve control performance. The methods involved in validating the control performance using the combined DNN are presented in Section 2.4.2.

The platform used for this work is a continuum joint comprised of four pressurized bellows which encircle an inextensible steel cable, as shown in Figure 2.2. Controlling the pressure in each of the bellows results in a net torque which causes the joint to bend. We use the same singularity-free kinematic relationships derived by [6] where the curvature of the continuum joint
is parameterized as two separate rotations ($u$ and $v$) about orthogonal axes ($x$ and $y$), which lie at the base of the joint. For notational clarity in this chapter, we define $\theta = u$ and $\phi = v$.

The dynamic model of the continuum joint is of the form

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = \tau$$

where $M(q) \in \mathbb{R}^{2\times2}$ is the symmetric mass matrix, $C(q, \dot{q}) \in \mathbb{R}^{2\times2}$ is the Coriolis matrix, $g(q) \in \mathbb{R}^2$ is a vector of torques caused by gravity, $q(t) = [\theta, \phi]^T$ is a vector of generalized coordinates, and $\tau \in \mathbb{R}^2$ is a vector of generalized forces.

An analytical equation of motion of the form shown in Equation 3.14 can be derived using principles of Lagrangian mechanics by modeling the joint as an infinite set of infinitesimally thin disks and integrating along the length of a piecewise constant curvature (PCC) arc. This method was developed in [29], which includes a detailed derivation of this model.

There are also significant nonlinear pressure dynamics inside of the bellow actuators, where the rate of change in pressures is on the same order of time response as the actual motion of the
robot. We model the pressure dynamics as a first-order system such that

\[ \dot{p}(t) = \alpha(p_{\text{ref}}(t) - p(t)) \] (2.2)

where \( p(t) \in \mathbb{R}^4 \) is a vector of pressures, \( p_{\text{ref}}(t) \in \mathbb{R}^4 \) is a vector of reference (i.e., commanded) pressures, and \( \alpha \in \mathbb{R}^{4 \times 4} \) is a diagonal matrix of coefficients representing the fill/vent rate of the pneumatic valves. Numerical values for the parameters used in this model are included in the repository accompanying the original journal paper [30].

Because each of the pressure bellows is made of deformable plastic, there are several effects from material properties such as stiffness and damping that are not accounted for in Equation 3.14. We include these effects as a linear spring term \((K_{\text{spring}}q, \text{where } K_{\text{spring}} \text{ is a diagonal matrix})\), which pulls the joint towards a completely vertical configuration, and a viscous damping term \((K_d \dot{q}, \text{where } K_d \text{ is also a diagonal matrix})\). The pressure-to-torque mapping term \((K_{\text{prs}}p)\) maps pressure differentials in each antagonistic pair of bellows to a torque about each axis where bending in \( \phi \) and \( \theta \) occur. These additions, coupled with Equation 2.2, result in our final analytical dynamic model:

\[ M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = K_{\text{prs}}p - K_d \dot{q} - K_{\text{spring}}q \] (2.3)

For conciseness, we rearrange Equations 2.2 and 2.3 into a nonlinear state variable form

\[
\dot{x}(t) = \begin{pmatrix}
-\alpha & 0 & 0 \\
M^{-1}K_{\text{prs}} & M^{-1}(-K_d - C) & -M^{-1}K_{\text{spring}} \\
0 & I & 0
\end{pmatrix}
\begin{pmatrix}
p \\ \dot{q} \\ q
\end{pmatrix}
+ \begin{pmatrix}
\alpha \\ 0 \\ 0
\end{pmatrix}
p_{\text{ref}}(t) - M^{-1}g
\] (2.4)

where \( x(t) \equiv [p, \dot{q}, q]^\top \) and \( u(t) \equiv p_{\text{ref}}(t) \). We use \( x(t) \) and \( u(t) \) for the remainder of this work.

### 2.3.3 Surrogate and Error DNN Training

With accurate dynamic equations, training a DNN approximation of modeled dynamics becomes a straightforward nonlinear function approximation problem. From the journal paper,
our collaborator Taylor Sorensen used data generated from the dynamic equations as presented in Section 2.3.2 to learn a DNN model. However, even with a near-perfect approximation of the analytical model of the robot, the DNN (labeled as $N_{\text{sim}}$) does not account for unmodeled hardware dynamics. Specifically, the pressure response was assumed to be first order, plastic deformation in the soft robot’s chambers cause the robot to have a constant offset in $\phi$ and $\theta$, and fluid flow may be choked by the robot’s valves. These assumptions limit the dynamic equations’ ability to accurately represent the hardware performance, but would be very difficult to model while maintaining tractability.

To approximate these unmodeled dynamics, data were gathered from the hardware system and used to train a second DNN (labeled as $N_{\text{err}}$) that approximates the error between $N_{\text{sim}}$ and the hardware’s actual response. These two DNNs were used in concert as shown in Figure 2.3 to perform the hardware experiments found in Section 2.4.

![Control diagram for running NEMPC in conjunction with the learned error model.](image)

Figure 2.3: Control diagram for running NEMPC in conjunction with the learned error model. $u^*$ indicates the optimal input chosen by the controller. This input is sent to the embedded pressure controller and we measure pressures $p$ and positions $q$ directly, while estimating $\dot{q}$.

By combining both DNNs, we obtained a function that can model the real hardware dynamics of the robot with surprising accuracy. This DNN learned to approximate choked air flow
(see Figure 2.4), a time-shift in velocity due to the estimated velocity state (see Figure 2.5), and joint angle equilibrium offsets due to the robot’s plasticity (see Figure 2.6). Extensive details of the process and results of performance in more scenarios can be found in the original work [30].

![Figure 2.4: Comparison of pressure dynamics between the four different systems used. The dashed line indicates the commanded pressure, while each of the solid lines is the pressure response resulting from the commanded pressure input. Note that the states from the analytical model and the surrogate DNN match well and that when using the combined DNN, the simulation closely resembles the hardware data.](image)

2.4 NEMPC Control Performance

In this section, we present our control algorithm and our findings based on several experiments in simulation and on hardware.

2.4.1 Nonlinear Evolutionary Model Predictive Control

Nonlinear evolutionary model predictive control (NEMPC) was developed as a real-time control algorithm for high degree of freedom (DoF) robot platforms. A variant of model predictive control (MPC), NEMPC utilizes an evolutionary algorithm to solve the MPC optimization. By
using an evolutionary algorithm, it is able to approximate a global minimum (as opposed to an exact local minimum) because it explores more of the solution space than local optimization methods. Extensive implementation details can be found in papers by [21, 23].

The implementation of NEMPC in this work differs from the work in [21] in that the algorithm no longer mutates every child generated during mating. With some probability $P_{\text{mutate}}$, children are selected for mutation. Those children have each of their genes perturbed by a uniform distribution on the interval $(-\sigma, \sigma)$. This allows the search to refine individual trajectories while still preserving others.

For this chapter, we implement the typical quadratic cost function formulation used in other MPC schemes with one small modification that places a cost on the change in inputs (i.e. $\Delta u_t = u_t - u_{t-1}$) as opposed to $u_t$ itself. This forces NEMPC to generate more conservative solutions which in turn, cause pressure to vary more smoothly over time. Note that the cost on change in inputs is a competing optimization objective with position tracking and requires some tuning of $Q$. 

Figure 2.5: Comparison of velocity dynamics between the four different models used. These velocities are the response resulting from the commanded pressure inputs shown in Figure 2.4. Note that the surrogate DNN tracks the analytical model well while the combined DNN tracks the hardware data well.
Figure 2.6: Comparison of joint angle dynamics between the four different models used. These angles are the response resulting from the commanded pressure inputs shown in Figure 2.4 and $R$ to achieve good tracking performance while also maintaining smooth input trajectories. The optimization is formulated as

$$
\text{minimize } J = \sum_{t=0}^{T-1} \left[ (x_t - x_{\text{goal}})^\top Q (x_t - x_{\text{goal}}) + \Delta u_t^\top R \Delta u_t \right] + (x_T - x_{\text{goal}})^\top Q_f (x_T - x_{\text{goal}})
$$

w.r.t. $u_t$, \( \forall t \in 0, 1, \ldots, T \)

s.t. $x_{\text{min}} \leq x_t \leq x_{\text{max}}, \ \forall t \in 0, 1, \ldots, T$

$u_{\text{min}} \leq u_t \leq u_{\text{max}}, \ \forall t \in 0, 1, \ldots, T$

$x_{t+1} = x_t + N, \ \forall t \in 0, 1, \ldots, T$

where

$$
N = N_{\text{sim}}(x_t, u_t)
$$
or

\[ N = N_{\text{sim}}(x_t, u_t) + N_{\text{err}}(x_t, u_t). \]  

(2.7)

In Equation 2.5, \( J \) is a scalar representing the cost of a given input sequence, \( T \) is the simulation horizon over which that input series is applied, and \( Q \in \mathbb{R}^{8 \times 8}, Q_f \in \mathbb{R}^{8 \times 8} \) and \( R \in \mathbb{R}^{4 \times 4} \) are diagonal weighting matrices penalizing error, error at the final time step of the horizon, and actuator effort, respectively. \( x_t \) represents the state vector and \( u_t \) is the input vector. \( x_{\text{goal}} \) is the commanded robot state. \( Q \) and \( Q_f \) are weighted such that the only values of \( x_{\text{goal}} \) that contribute to the cost \( J \) are the position and velocity states. The variable \( N \) is a placeholder for the DNN that NEMPC uses. For the case using the surrogate DNN defined in Section 2.3.3, NEMPC enforces the constraint given in Equation 2.6. For the combined case defined, NEMPC uses the constraint given in Equation 2.7.

At each time step, the optimizer is allowed to take a single step towards the optimum (or one generation of the genetic algorithm). NEMPC then returns the input associated with the lowest cost member of the population for the current time step, which is applied to the hardware system. As soon as that command is sent, NEMPC takes another step towards the optimum, given new measurements of the robot’s state. The fact that the previous time step’s population is used to warm start the next optimization causes the algorithm to converge quickly.

As a practical note, the tuned weights in \( Q \) corresponding to the pressure states are 0 because we are not trying to follow a pressure trajectory or specify stiffness. This allows NEMPC to find any valid set of pressure states that will enable tracking of desired velocity and positions. Positions are weighted heavily and velocities relatively lightly.

The introduction of a DNN as NEMPC’s internal model of the plant is a key component that enables NEMPC’s execution at real-time speed and the evaluation of an entire population of solutions in batches. This allows a large graphics processing unit (GPU) to simultaneously evaluate all 1500 potential input series at any given time step. In our work, we are able to control the 8 state soft robot continuum joint at a rate of 100 Hz with a time horizon of 0.1s.
2.4.2 Hardware Experiments

For our experiments in hardware, we evaluated the performance of NEMPC while controlling the soft robot continuum joint, following a reference trajectory in $\theta$ and $\phi$. This experiment is run twice, once while NEMPC’s internal model of the robot is represented by the surrogate DNN ($N_{\text{sim}}$), and once while NEMPC’s internal model is represented by the combined DNN ($N_{\text{sim}} + N_{\text{err}}$).

We use two HTC Vive Trackers rigidly attached to the robot base and tip in order to measure joint angles ($\theta$ and $\phi$) in real-time (see Figure 2.7), while the joint velocities ($\dot{\theta}$ and $\dot{\phi}$) are numerically estimated from the angle measurements. The pressures in each of the robot’s four chambers are measured by onboard sensors and controlled by an embedded high-frequency PID controller. All of this data is packaged and published via the Robot Operating System (ROS) at 400 Hz to a separate computer on the network with an 8 core Intel Xeon E5-1620 CPU and an NVIDIA GeForce GTX 1080 Ti GPU, which is dedicated to running the NEMPC algorithm. As shown by [31], the hardware requirements for major deep learning papers have increased quickly with time, so we believe that our single-GPU setup is relatively inexpensive and computationally cheap.

![Diagram of the experimental setup for the hardware experiments. Also illustrated here is the inherent plasticity of the robot, resulting in a variable offset in $\theta$ and $\phi$. Over time, the plastic in the pressure chambers deforms and causes the robot to have an equilibrium configuration that is not vertical.](image-url)
Figure 2.3 illustrates the process as a control diagram. The controller is given an \( x_{\text{des}}(t) \) which is used in conjunction with the current state estimate \( \hat{x}_t \) to calculate an optimal pressure command \( u^* \). This command is sent to the embedded PID pressure controller and then pressures and joint angles are measured directly.

### 2.4.3 Hardware Results

The results of the hardware experiments are presented in Figure 2.8. When the surrogate DNN is used as NEMPC’s internal model to control the soft robot hardware, NEMPC struggles to follow the desired path for \( \theta \) and \( \phi \). This behavior is likely due to the surrogate DNN’s poor approximation of the hardware dynamics, as evaluated in Section 2.3.3. Evidence of this is found in the performance of NEMPC while internally simulating with the combined DNN.

![Figure 2.8: Comparison of tracking performance on the physical soft robot continuum joint while using the two categories of DNN model approximation. Note that the control performance of NEMPC while using the combined DNN contains much less steady-state tracking error than the control performance of NEMPC while using the surrogate DNN.](image)

When NEMPC controls the hardware while using the combined DNN as its internal model, the reference tracking performance shown in Figure 2.8 improves significantly. With a more ac-
curate internal model, NEMPC is able to generate solutions that better account for factors such as the robot’s plasticity (e.g., non-zero equilibrium configuration), hysteresis, and increased stiffness and damping near joint limits. This results in a much lower steady-state offset, and more rapid convergence in some cases.

Quantitatively, the reference tracking behavior of NEMPC can be measured through a statistical analysis of the tracking error for each experiment. A statistical comparison of NEMPC performance can be found in Table 2.2. The mean tracking error decreased from 0.378 rad to 0.182 rad, a 52% decrease. The median tracking error decreased by almost an order of magnitude. Of particular note is the difference in integral of the time-weighted absolute error (ITAE) for each trial. This measure penalizes errors that persist over time, and allows a controller to be slightly less aggressive, as long as it converges and stays close to its target. The ITAE is calculated for each step input individually, summed over the whole series of step inputs, then recorded. As seen in the table, NEMPC with the combined DNN greatly outperforms NEMPC with the surrogate DNN in regards to ITAE, in part due to its lack of significant steady-state error. The surrogate DNN could be helped by the addition of an integrator to the controller, as done in previous work with NEMPC by [21]. In Table 2.2, the execution time for a single time step is listed in seconds as well as a multiplier of many times faster the DNN execution time was compared the analytical model implemented in C++.

Table 2.2: Comparison of control performance of NEMPC with error compensation versus NEMPC without error compensation.

<table>
<thead>
<tr>
<th></th>
<th>ITAE</th>
<th>Mean Tracking Error</th>
<th>Median Tracking Error</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_{sim}$</td>
<td>128.3496 rad$^2$s</td>
<td>0.37820 rad</td>
<td>0.31736 rad</td>
<td>.0006 s (464x)</td>
</tr>
<tr>
<td>$N_{sim} + N_{err}$</td>
<td>21.5252 rad$^2$s</td>
<td>0.18180 rad</td>
<td>0.03676 rad</td>
<td>.0009 s (287x)</td>
</tr>
</tbody>
</table>

What is most impressive in this case is that by incorporating the combined DNN with NEMPC, we achieve very low steady-state error with no integral control at all. All of our prior work (and most of the soft robot control literature) has required some sort of integral or adaptive control to compensate for this steady-state error (see [29] for an example of model-reference adaptive control (MRAC) which essentially exhibits integral action to achieve low steady-state error).
The implementation of an integrator could help reduce steady-state tracking error for the surrogate DNN controller, but the control would still suffer from overshoot and generally poor performance. The mean, median, and standard deviation of the tracking error would likely remain indicators of the surrogate DNN’s relatively poor performance. To visualize the insights offered by the mean, median, and standard deviation of the tracking error, Figure 2.9 presents a histogram of the normalized frequency of error for each of the two experiments on hardware. Visible in the plot for the surrogate DNN is the angle offset due to the robot’s non-zero equilibrium configuration. The surrogate DNN causes NEMPC to tend towards negative error in $\theta$ and positive error in $\phi$. When the error model in the combined DNN is introduced, both $\theta$ and $\phi$ error are pulled towards zero, becoming uni-modal and more normally distributed. Overall, the combined DNN is a much better approximation of the robot’s dynamics, allowing NEMPC to follow the given reference trajectory much more effectively, even with fast changes (step inputs) in the commanded changes for $\phi$ and $\theta$.

Figure 2.9: A histogram of the normalized frequency of $\theta$ and $\phi$ tracking error in the hardware experiments. Note that the data gathered while using the surrogate DNN for control has $\theta$ error and $\phi$ error that is biased in both directions away from zero. This is a result of the surrogate DNN’s lack of information regarding the offsets in $\theta$ and $\phi$ at equilibrium. Also note the difference in y axis scaling for both histograms.
To validate that the combined DNN can be used for control trajectories other than step inputs, we conducted two more experiments: one for tracking sin waves in $\phi$ and $\theta$ and a second for tracking ramps in $\phi$ and $\theta$. The results can be seen in Figure 2.10. From these figures, it is apparent that the training data consisting of only step inputs is enough for the DNN to accurately predict the performance of the robot while tracking other wave forms. There is a nominal amount of phase lag in both cases, but this is expected because, in our implementation of NEMPC, $x_{\text{goal}}$ for the entire prediction horizon remains constant while the waveform continuously changes. This could be overcome (without changing our formulation at all) by simply allowing NEMPC to use a continuous $x_{\text{goal}}$ trajectory instead of a single constant value which we used.

![Figure 2.10: Comparison of tracking performance on sine (left column) and ramp (right column) test signals using the $N_{\text{sim}} + N_{\text{err}}$ DNN configuration for control. Note that although both DNNs were trained solely on step inputs, the models are able to generalize well to other types of signals.](image)

### 2.5 Conclusion

In this work we demonstrate that significant model and control improvement is possible through a data-driven deep learning approach. Our approach does not require specialized expertise
or any assumptions about the form of the model. As a result, this method is generally applicable to many model-based control problems where the plant dynamics are highly uncertain or only partially known.

Additionally, because our approach is rooted in a physics-based analytical model and our error DNN only needs to learn relatively small adjustments, the error DNN can be smaller, faster, and train with less data than would be required if we took a completely model-free learning approach. This is especially beneficial when gathering training data on hardware is dangerous or expensive, as is often the case in the field of robotics (albeit less so for many soft robots).

An important preliminary result, though not discussed in-depth in this chapter, is that the model and controller were sensitive to the frequency content in the data used for training. The effects of this were significant, but are currently poorly understood. However, we have presented evidence that using square waves to explore and learn the state space is an efficient method because the trained models generalized relatively well to sine waves and ramps. We also note from our experiments that the inverse relationship is not true; models trained on sine waves and ramps generally did not perform well when tested on step inputs. We believe this is because square waves excite more dynamic modes than sine waves or ramp inputs in pressure.

With a better model of hardware dynamics, NEMPC and other predictive control algorithms can more accurately predict how a system will respond over time. In Chapter 3, we explore the ability of NEMPC and particle swarm model predictive control (PSOMPC) to control robotic systems with inherent uncertainty, additionally evaluating NEMPC and PSOMPC performance on the same soft robot used in this chapter.
CHAPTER 3. OPTIMAL CONTROL FOR NONLINEAR ROBOTIC SYSTEMS IN THE PRESENCE OF UNCERTAINTY

3.1 Motivation

Up to this point, NEMPC has been exclusively used under the assumption that its internal model of the plant is representative of the real plant dynamics. Chapter 2 demonstrated the poor performance of NEMPC when modeling error was present, and focused on using a deep neural net (DNN) to close the modeling error gap. In this chapter, we focus on designing a better controller to operate under conditions of uncertainty, as would be expected in real-world scenarios such as search and rescue operations.

This chapter considers three forms of uncertainty: parameter uncertainty, input disturbances, and state measurement noise. NEMPC performance as well as performance of a proposed particle swarm model predictive control (PSOMPC) algorithm is demonstrated on two simulated systems and a soft robot continuum joint in simulation and hardware. The remainder of this chapter is a journal paper in preparation, of which I am the primary author.

3.2 Introduction

Robotic systems are being used for complex tasks previously thought to be only within the capabilities of humans. With better control, more advanced sensors, and faster on-board computers, robots are beginning to operate outside the realm of what was previously possible. However, only so much can be accomplished by robots that must operate within a structured lab environment. Future innovation will require robots and algorithms to be robust enough to operate in uncertain environments such as locomoting across uneven terrain, handling variable payload situations, and operating in close proximity with humans. As we develop these robots, we must also consider how to address this additional uncertainty in their control.
In this work, we investigate a method of incorporating uncertainty modeling into a non-linear model predictive control (MPC) algorithm, based on the nonlinear evolutionary model predictive control (NEMPC) algorithm developed by Hyatt et al. [2]. We develop a novel implementation of a parallelized sampling-based controller using a particle swarm optimization. Control performance of a proposed particle swarm model predictive control (PSOMPC) algorithm with uncertainty modeling is compared to NEMPC control performance.

Hyatt et al. developed NEMPC as an algorithm to execute quickly and account for the nonlinear dynamics of high DoF systems such as soft robots. However, the formulation, which uses a genetic algorithm, does not allow us to make assumptions about relationships between population members or maintain multiple populations to track different solutions as they change over time. If we could make assumptions about the relationships between population members and track multiple candidate solutions simultaneously, we could choose an optimal input series that is more robust to disturbances, modeling error, and estimation error. For the purpose of defining a relationship between population members and tracking multiple solutions, this chapter introduces particle swarm model predictive control (PSOMPC), a variant of NEMPC that utilizes the particle swarm (PSO) algorithm and takes advantage of its multi-swarm capabilities and mathematical particle update to ensure better exploration of potential solutions.

The structure of the remainder of the paper is described next. In Section 3.3 we first explore related work in the fields of optimal control, heuristic optimization, and control for uncertainty. In Section 3.4 we present a heuristic model predictive control algorithm as a basis for both PSOMPC and NEMPC. In Section 3.5 we compare NEMPC and PSOMPC performance on two conventional simulated systems with introduced uncertainty conditions. Then we demonstrate both algorithms’ performance on a continuum soft robot joint in Section 3.6 in simulation and hardware. Due to the discontinuity introduced as part of the control cost function (introduced in section 3.4), we are unable to compare PSOMPC and NEMPC performance directly to previous methods, but in [2], a comparison of NEMPC and conventional optimal control methods can be used as a baseline for NEMPC performance. Throughout the paper in all mathematical formulas, bold lowercase letters indicate vectors, while bold uppercase letters indicate matrices.
3.3 Related Work

Many advanced algorithms such as model predictive control (MPC), reinforcement learning (RL), and model reference adaptive control (MRAC) have been used successfully used in high degree-of-freedom (DoF) robot control. However, many of these algorithms require significant assumptions about the system to be controlled. Many formulations of MPC require a quadratic cost function and see rapid performance degradation as modeling error increases [1, 32]. RL often requires representative data, a nominal control strategy, and struggles with extrapolating the policy to scenarios with uncertainty in the state estimate, model parameters, or input disturbances.

Of particular note are MPC formulations that utilize heuristic optimization strategies as heuristic algorithms enable MPC to account for constrained optimization, as well as non-quadratic cost functions (see [1, 2]). Genetic algorithms (GA) and particle swarm optimization (PSO) are two widely used heuristic-based optimization algorithms [33–35], and have been used to solve the MPC problem with good results. Many of these formulations utilize particle swarm optimization (PSO) [33, 36–41], while others utilize a genetic algorithm (GA) [2, 34, 42], and several apply variants of stochastic algorithms [43–46].

It has been shown that, while standard gradient-based optimization solution methods have superior performance in quadratic cost landscapes, heuristic algorithms prove more effective in multi-modal or discontinuous situations [47]. In the case of robotic control, the cost functions may be far from quadratic in form and the system dynamics may be nonlinear, making heuristic optimization a good choice for real-time optimal control. In addition, heuristic optimizations in MPC have the ability to search for nonlocal solutions, not depending on initialization in the neighborhood of a local minimum or nominal controller [2, 38]. This allows the MPC algorithm to have a better chance of finding a global optimum which may not be intuitive to a human operator or conventional optimal control algorithm.

Heuristic optimizations also lend themselves to straightforward parallelization through use of multiple CPU cores [40], field programmable gate arrays (FPGA) [39, 48], or graphics processing units (GPU) [2, 38, 42–45, 49]. This allows them to execute faster for large populations.

Non-parallelized implementations of MPC with complex dynamics, costs, constraints, or long time horizons often do not run fast enough for real-time control of hardware systems [1], except where the system’s response is very slow [36]. Parallelized implementations, however,
have been able to reach control speeds suitable for low-DoF systems such as an acrobot [38] and a miniaturized auto-rally car [44]. Still, most systems require short time horizons and small PSO or GA populations in order to evaluate at a rate appropriate for real-time control.

For our formulation of nonlinear MPC, a deep neural net is trained to approximate system dynamics, and is then used as the controller’s internal model of the plant. This allows the cost function of the algorithm to be executed in parallel, greatly reducing computation time, and allowing for large populations in the GA and PSO-based solution methods, while also allowing for the approximation of unmodeled hardware dynamics [30].

In addition to speeding up algorithm evaluation through parallelization, NEMPC and PSOMPC formulate the trajectory optimization as a dynamic optimization. Most of the formulations of PSO and GA for MPC are static optimizations, where the cost function remains fixed over time. This requires initializing a population and running the algorithm to completion every time an optimal input is desired. As a result, GA and PSO populations sizes would need to be very small to achieve real-time control speeds for high-DoF systems. NEMPC and PSOMPC are two of the few that utilize a dynamic optimization, where the cost function changes over time and optima shift and change at each time step. Dynamic optimization allows for faster execution, but requires a different formulation of the optimization, configuring the algorithm to track moving peaks and valleys over time and continually identifying which peak or valley is the global optimum. GA and PSO are often used in this scenario [35].

NEMPC and PSOMPC make this dynamic optimization more tractable by parameterizing the MPC input trajectory as a number of knot points, instead of an independent input applied at each time step of the MPC horizon [1]. This reduces the dimensionality of the optimization to three or four parameters per input, even for long time horizons. This has enabled the control of high degree-of-freedom (DoF) systems without specialized assumptions, including a 24 state soft robot arm at a rate of 372Hz [42].

Given more efficient algorithms to achieve real-time control of high-DoF robots, uncertainty in the system and environment must also be taken into account for these systems to operate effectively outside of a controlled lab environment. Robust control techniques have been used extensively to control systems within defined uncertainty bounds [50, 51], but they account for unmodeled uncertainty by reducing control gains to remain far from the limits of control perfor-
mance. If uncertainty could be modeled and accounted for, predictive control techniques could still push the system to the limits of its performance while remaining robust to uncertainty where uncertainty poses a risk of causing the control inputs to violate system constraints.

Several approaches have been used to model uncertainty and incorporate that model in the controller [50,52]. Significant research using this method has been applied to the control of aircraft in the presence of wind disturbances. The experience-driven predictive control algorithm has been developed extensively [53–57] and uses principles from MPC and Gaussian process-based MRAC to learn various controllers online. Various methods have required classifying the uncertainty in the system [58,59], while other methods learn parameters of the system [60]. Several forms of reinforcement and online learning have also been applied to the problem, with excellent results [61]. The uncertainty modeling explored in the PSOMPC algorithm in Section 3.4.2 presents an approach to accounting for uncertainty in control by introducing noise into the cost function evaluation, and tracking multiple groupings of candidate solutions simultaneously. To understand these methods, we must first present the nonlinear heuristic model predictive control structure.

### 3.4 Nonlinear Heuristic Model Predictive Control Formulations

The two forms of MPC used in this work can be described as variants of the same MPC formulation: nonlinear heuristic model predictive control (NHMPC). NHMPC gives three benefits over traditional MPC formulations: speed of execution, compatibility with any arbitrary cost function, and compatibility with DNN approximations of plant dynamics for model-based prediction across the given time horizon.

#### Execution Speed

NHMPC reduces computation time by formulating the MPC optimization as a dynamic optimization, where the given optimization algorithm approaches and follows valleys that shift in the optimization space at each time step, sometimes referred to as the moving peaks problem [62]. Heuristic optimization algorithms have shown good performance on the moving peaks problem, assuming they are configured correctly [35,63,64]. Formulating the optimization in this manner negates the need to run a full optimization for every control input applied to the robot, greatly
decreasing the algorithm’s computation time. NHMPC formulations have been shown to execute very quickly, even when compared to MPC formulations with convex solvers [1].

**Arbitrary Cost Function**

Since NHMPC requires no gradients and makes no assumptions about the shape of the cost landscape, cost functions used with NHMPC can be nonlinear, non-quadratic, and can contain sharp nonlinearities and discontinuities that would cause other optimization routines to fail. In this work, the cost function used is similar in form to the typical quadratic cost function of LQR, with the addition of a penalty for obstacle collisions, as shown in Equation 3.1.

**DNN Compatibility**

NHMPC utilizes a DNN approximation of system dynamics in evaluating the cost of a given solution. By doing so, NHMPC can evaluate an entire population of solutions simultaneously on a graphics processing unit (GPU), speeding up objective evaluation dramatically. This work uses the same prediction methods as [30], and approximates all simulated and hardware systems using the same DNN structure. Specific hyperparameters and training code are available upon request from the authors.

\[
\begin{align*}
\min_{u_t} \quad & J = \sum_{t=0}^{T} \left[ \lambda_t \beta + (z_t - z_{\text{goal}})^\top Q(z_t - z_{\text{goal}}) \right] + \Delta u_0^\top R \Delta u_0 + (z_T - z_{\text{goal}})^\top Q_f (z_T - z_{\text{goal}}) \\
\text{w.r.t.} \quad & u_t, \quad \forall t \in 0, 1, \ldots, T \\
\text{s.t.} \quad & z_{\text{min}} \leq z_t \leq z_{\text{max}}, \quad \forall t \in 0, 1, \ldots, T \\
& u_{\text{min}} \leq u_t \leq u_{\text{max}}, \quad \forall t \in 0, 1, \ldots, T \\
& z_{t+1} = z_t + N(z_t, u_t), \quad \forall t \in 0, 1, \ldots, T \\
& \lambda_t = \begin{cases} 
1 & \text{if } z_t \in \mathcal{X} \\
0 & \text{otherwise} 
\end{cases}, \quad \forall t \in 0, 1, \ldots, T 
\end{align*}
\]  
(3.1)
In Equation 3.1, \( J \) is a scalar representing the cost of a given input sequence, \( T \) is the simulation horizon over which that input series is applied, and \( Q, Q_f \) and \( R \) are diagonal weighting matrices penalizing error, error at the final time step of the horizon, and actuator effort, respectively. \( z_t \) represents a vector used to penalize state error (most often \( z_t \) is the system’s state vector) and \( u_t \) is the input vector. \( z_{\text{goal}} \) is the commanded \( z \). \( Z \) defines the region of the obstacle, such that when \( z \in Z \), the system is in collision with the obstacle. \( \beta \) is the collision penalty weighting, presented for each system. \( z_t, z_{\text{goal}}, Z, \) and \( \beta \) are defined for each system. The variable \( N \) is a placeholder for the DNN that NHMPC uses as its internal model of the plant, and is trained to approximate the system’s state transition function.

For the purposes of obstacle avoidance, it may be possible to formulate a much better cost function. However, such an exploration merits its own work as cost function formulation is far from trivial, and can affect system performance in significant and non-intuitive ways. However, the cost function presented in Equation 3.1 is well suited for this work as it encourages the system to avoid collisions, but does nothing to encourage the system to distance itself from obstacles. As will be shown in Section 3.5, any distance maintained from the obstacle must be attributed to a controller’s robustness to uncertainty, and will allow us to compare the efficacy of the control algorithm without the effects of cost function formulation.

Algorithm 3 presents the basic algorithm for NHMPC. \( P \) represents the entire population of solutions in a normalized optimization space. Each member of \( P \) represents \( m \) inputs across a time horizon parameterized by \( k \) knot points, as presented in [2]. The knot point parameterization has been shown to make the optimization more tractable by reducing the dimensionality of the search space without significantly affecting the quality of the solutions found. \( \mathcal{U} \) is the uniform distribution, and \( J \) is the cost of a given input trajectory. \( T \) represents the time horizon over which MPC will evaluate cost, and \( \text{get}\_u\_from\_P \) interpolates the parameterized trajectories and maps the solution from optimization space (\( p \)) to system input space (\( u \)). \( N \) represents the DNN approximation of the plant’s dynamics, which returns a \( \Delta x \) to be added to \( x_t \). \( \text{get}_\text{new}_\text{generation} \) performs one step of the dynamic optimization and is defined in Algorithms 4 and 5 for NEMPC and PSOMPC, respectively. \( P^* \) represents the best solution currently located by the optimization, and is retrieved by \( \text{get}_\text{best}_\text{P} \), which returns the population member with the lowest cost. We next describe the specific implementations of NEMPC and PSOMPC used in this paper.
Algorithm 3 Nonlinear Heuristic Model Predictive Control Algorithm

1: for every simulation in parallel do
2:    if Cold Start then
3:      \( P = \mathcal{U}(p_{\text{min}}, p_{\text{max}}) \)
4:    else if Warm Start then
5:      \( P = \text{get new generation}(P, J) \)
6:    end if
7:  \( J = 0 \)
8:  for \( t = 0 \) to \( T \) do
9:    \( u_t = \text{get u from } P(P, t, T) \)
10:   \( x_{t+1} = x_t + N(x_t, u_t) \)
11:  \( J = J + \text{cost function}(x_t, u_t, t) \)
12:  end for
13: end for
14: \( P^* = \text{get best } P(P) \)
15: \( u^* = \text{get u from } P(P^*, 0, T) \)
16: Apply \( u^* \) to the robot

3.4.1 NEMPC

The first variant of NHMPC is nonlinear evolutionary model predictive control (NEMPC), which has been presented and evaluated in several different forms [2, 30]. In this work, NEMPC is represented as a variant of NHMPC, with a generational update as shown in Algorithm 4.

Algorithm 4 Nonlinear Evolutionary Model Predictive Control Generation Update

1: \( P_{\text{parents}} = P\text{’s with lowest } J\text{’s} \)
2: for every child in parallel do
3:    Randomly select two parents from \( P_{\text{parents}} \)
4:    \( P_{\text{child}} = \text{crossover}(P_{\text{parent1}}, P_{\text{parent2}}) \)
5:    if \( \mathcal{U}(0, 1) < p_{\text{mutate}} \) then
6:        \( \sigma = \sigma_{\text{noise}} e \)
7:        \( P_{\text{child}} = P_{\text{child}} + \mathcal{N}(\mu = 0, \sigma) \)
8:    end if
9: end for
10: \( P_{\text{strangers}} = \mathcal{U}(p_{\text{min}}, p_{\text{max}}) \)
11: \( P = \text{concatenate}(P_{\text{parents}}, P_{\text{strangers}}, P_{\text{children}}) \)

In Algorithm 4, the optimizer performs a genetic algorithm update to obtain the next generation of solutions. \textit{crossover} constructs a \( P_{\text{child}} \) by taking individual real valued genes from \( P_{\text{parent1}} \).
or $P_{\text{parent2}}$ with probability 0.5. $p_{\text{mutate}} \in (0, 1)$ represents the probability of a given child being mutated. $\sigma_{\text{noise}}$ is a hyperparameter that controls the magnitude of mutation noise applied to mutated children. $\sigma$ is the noise applied at a given time step, and decays to zero as tracking error decays to zero. $e$ is a measure of tracking error that is defined with each individual experiment. $N$ is the normal distribution, and concatenate concatenates sub-populations into one full population. $P_{\text{strangers}}$ are randomized trajectories introduced to the population at every time step for the purpose of adding potentially good genetic information to the gene pool, $p_{\text{min}}$ and $p_{\text{max}}$ represent the maximum and minimum values in the normalized search space. For this work $p_{\text{min}} = -100$, $p_{\text{max}} = 100$.

Table 3.1 presents the tunable hyperparameters for NEMPC. For each of the software experiments, baseline hyperparameters were identified by the Optuna library [65] with 1000 iterations. The objective was defined as the integrated time-absolute error summed over three two-second control rollouts on a system with parameter uncertainty, state measurement noise, and a wandering input disturbance. These uncertainty conditions are defined in the hyperparameters table for each system in Sections 3.5 and 3.6. Three two-second rollouts is likely not enough to get a statistically significant measure of performance, but for the purpose of identifying just one set of valid hyperparameters, it was determined to be sufficient. An identical process was used for determining the PSOMPC hyperparameters presented in Table 3.2. This method of obtaining hyperparameters is by no means comprehensive, and future work should explore the hyperparameter space of each algorithm and analyze their effects on control performance, particularly for PSOMPC.

### 3.4.2 PSOMPC

Particle swarm model predictive control (PSOMPC) and its uncertainty robustness modifications are the main contribution of this work. NEMPC has already proven to be effective in general control [2] and in control of soft robot arms [30]; however, it only maintains a single population of input trajectories, and its combinatorial nature inhibits any assumptions on the relationships between population members. Particle swarm optimization, however, can maintain multiple swarms of particles, with each swarm tracking one locally optimal solution and particles within a swarm interacting and sharing information. This allows PSOMPC to find better solutions, and allows us to introduce noise on each particle to roughly model the uncertainty in our system.
Table 3.1: Hyperparameters for NEMPC Algorithm

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\text{noise}}$</td>
<td>The magnitude of mutation noise applied to mutated children. Decays to zero as tracking error approaches zero.</td>
</tr>
<tr>
<td>$n_{\text{parents}}$</td>
<td>The number of parents in a generation.</td>
</tr>
<tr>
<td>$n_{\text{strangers}}$</td>
<td>The number of random strangers to introduce to a population at every time step.</td>
</tr>
<tr>
<td>$p_{\text{mutate}}$</td>
<td>The probability that a given child will be mutated</td>
</tr>
</tbody>
</table>

In order to understand the relationships between particles in swarms, one must understand the fundamental principles of particle motion. Each particle in $P$ belongs to one of $n$ swarms, and is defined by a position $\mathbf{p}$ and velocity $\dot{\mathbf{p}}$ in the normalized optimization space. The acceleration in the optimization space of particle $i$ at a timestep $t$ is given by Equation 3.2. We will refer to ‘particle dynamics’ when discussing the PSO algorithm as a way of conceptualizing the optimization’s search process. It is important to note that terms such as ‘position,’ ‘velocity,’ and ‘repulsion’ in this case are merely constructs to help us understand swarm motion, and do not bear the same relationships as with conventional dynamics of the system to be controlled.

$$a_i(t) = c_p \varepsilon_p \cdot (\mathbf{b}_{p,i} - \mathbf{p}_i) + c_s \varepsilon_s \cdot (\mathbf{b}_s - \mathbf{p}_i) + \gamma \mathbf{a}_{q,i}$$ \hspace{1cm} (3.2)

$$c_s = \frac{e}{e_{\text{max}}} (c_{s,\text{max}} - c_{s,\text{min}}) + c_{s,\text{min}}$$ \hspace{1cm} (3.3)

In Equations 3.2 and 3.3, $c_p$ and $c_s$ represent the social and cognitive weighting, while $\varepsilon_p$ and $\varepsilon_s$ are drawn from $\mathcal{U}(0,1)$ at each evaluation of $a_i$. $\mathbf{b}_{p,i}$ and $\mathbf{b}_s$ represent the best location located by the particle itself and the swarm, respectively. $c_s$ decays from $c_{s,\text{max}}$ to $c_{s,\text{min}}$ as tracking error $e$ increases to $e_{\text{max}}$. $e$ is defined for each set of experiments individually. $\gamma$ is the Coulomb...
repulsive force scaling term, and $a_{qi}$ is the acceleration of the particle due to the Coulomb forces exerted on it by other charged particles, as defined in Equations 3.4 and 3.5.

$$a_{qi} = \sum_{j=k}^{h} \left[ \frac{-Q_i Q_j}{||r_{ij}||^3} \right] r_{ij} \tag{3.4}$$

$$r_{ij} = p_j - p_i \tag{3.5}$$

In Equations 3.4 and 3.5, $k$ and $h$ represent the indices in $P$ of the first and last particles in an independent swarm, and $Q$ the charge on a given particle. In this work, $50\%$ of the particles in each swarm are charged with $Q = 1.0$ and only interact with charged particles within their own swarm.

**Algorithm 5** Particle Swarm Model Predictive Control Generation Update

1: for each swarm do
2: \hspace{1em} $P_{\text{best}} = P$ in swarm with lowest $J$
3: \hspace{1em} for every particle in parallel do
4: \hspace{2em} set “a” according to eqs. 3.2 through 3.5
5: \hspace{2em} $\rho = e(\rho_{\text{max}} - \rho_{\text{min}}) + \rho_{\text{min}}$
6: \hspace{2em} $P(t + 1) = P(t) + \rho(x, x_{\text{goal}})v + a$
7: \hspace{1em} end for
8: end for
9: if $r_{i,j} < r_{\text{valley}}$ for any $i, j$ then
10: \hspace{1em} randomize the swarm with the highest swarm cost
11: end if
12: if $d_{\text{swarm}} < d_{\text{converge}}$ for all swarms then
13: \hspace{1em} randomize swarm with highest swarm cost
14: end if

Algorithm 5 presents the PSOMPC generational update, where $\rho$ is defined as the particle inertia term, which decays linearly from $\rho_{\text{max}}$ to $\rho_{\text{min}}$ as the tracking error of the system approaches 0. $r_{i,j}$ is the distance between swarm $i$ and swarm $j$, and $d_{\text{swarm}}$ is the swarm diameter, the maximum distance between any two particles in a swarm in any dimension. Lines 9-11 and 12-14 represent the exclusion and anti-convergence operators as presented in [62]. The exclusion operator prevents any two swarms from tracking the same local minimum, ensuring that each candidate optimum is tracked by only one swarm. The anti-convergence operator ensures that at least
one swarm is free to explore the solution space. Both $r_{\text{valley}}$ and $d_{\text{converge}}$ are hyperparameters for PSOMPC.

Table 3.2: Hyperparameters for PSOMPC Algorithm

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho_{\text{min}}$</td>
<td>The minimum particle inertia.</td>
</tr>
<tr>
<td>$\rho_{\text{max}}$</td>
<td>The maximum particle inertia</td>
</tr>
<tr>
<td>$c_{\text{sm}}$</td>
<td>The minimum social weight.</td>
</tr>
<tr>
<td>$c_{\text{sm}}$</td>
<td>The maximum social weight.</td>
</tr>
<tr>
<td>$n_{\text{swarms}}$</td>
<td>The number of independent swarms.</td>
</tr>
<tr>
<td>$r_{\text{valley}}$</td>
<td>The minimum distance within which two swarms can maintain a $b_z$.</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>A scaling factor for Coulomb repulsion between charged particles.</td>
</tr>
<tr>
<td>$d_{\text{converge}}$</td>
<td>The diameter at which a swarm is considered converged. The swarm diameter is the largest distance on any axis between any two particles in a swarm.</td>
</tr>
<tr>
<td>$\sigma_{\text{noise}}$</td>
<td>The magnitude of noise to apply to $u_i$ while forward simulating during cost function evaluation.</td>
</tr>
</tbody>
</table>

Table 3.2 presents the tunable hyperparameters for PSOMPC. By increasing the number of individual swarms with $n_{\text{swarms}}$ and introducing input noise on the simulation of each particle with $\sigma_{\text{noise}}$, PSOMPC will be able to roughly model the uncertainty in the plant. In addition, we can utilize the multiple swarms present as part of the PSO algorithm to track all candidate solutions and select the best global optimum. In the software experiments in Section 3.5, we will see the results of these effects on the performance of a mass-spring-damper and a simulated three-link robot arm.
3.5 Simulation Experiments

Simulation experiments were conducted to compare PSOMPC and NEMPC performance for the control of two conventional systems: a nonlinear mass-spring-damper, and a three-link robot arm in the presence of gravity. Both systems underwent trials in five different uncertainty scenarios, as presented in Table 3.3.

Table 3.3: Uncertainty Scenarios for Simulated Experiments

<table>
<thead>
<tr>
<th>Uncertainty Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>The plant is characterized by the nominal system parameters. No disturbances act on the system.</td>
</tr>
<tr>
<td>Parameter</td>
<td>The plant is characterized by parameters drawn from a uniform distribution between a defined max and min value. No disturbances act on the system.</td>
</tr>
<tr>
<td>Input</td>
<td>The plant is characterized by the nominal system parameters. Gaussian noise $u_{\text{noise}}$ with a wandering mean is added to the system inputs.</td>
</tr>
<tr>
<td>State</td>
<td>The plant is characterized by the nominal system parameters. Gaussian noise $x_{\text{noise}}$ is added to the measurement of the system’s state vector.</td>
</tr>
<tr>
<td>All</td>
<td>All three forms of uncertainty are present simultaneously.</td>
</tr>
</tbody>
</table>

As the uncertainty scenario changes for the plant, the internal simulation model for each controller remains constant. As model parameters are presented for each simulated system, parameters altered by each of the uncertainty conditions in Table 3.3 will be identified.
3.5.1 Nonlinear Mass-Spring-Damper

System Definition

The first simulated system is a nonlinear mass-spring-damper system, incorporating traditional mass-spring-damper dynamics, a stiffening ($\alpha > 0$) or softening ($\alpha < 0$) spring, and a friction model incorporating Stribeck friction, viscous friction, and both kinetic and static coulomb friction [66]. The mass-spring-damper is constrained by a wall located at $x_{\text{wall}} = 1.0$, as shown in Figure 3.1. A collision with this wall constitutes a violation of the collision cost as defined in the NHMPC cost function presented in Equation 3.1. The dynamics of the mass-spring-damper with input noise are presented in Equation 3.6.

![Diagram of the constrained nonlinear mass-spring-damper system](image)

Figure 3.1: The constrained nonlinear mass-spring-damper system, with a wall (barrier) located at $x_{\text{wall}} = 1.0$. The system is pictured at rest, with $x = 0$.

$$u + u_{\text{noise}} + F_f = m\ddot{x} + b\dot{x} + kx + \alpha x^3$$  \hspace{1cm} (3.6)

In Equation 3.6, $u$ represents the system input in Newtons, $u_{\text{noise}}$ represents the input disturbance present in some scenarios. $m$ represents the mass, $b$ the damping, and $k$ the linear spring
constant. $\alpha$ is the nonlinear spring coefficient, representing a stiffening spring for $\alpha > 0$ and a softening spring for $\alpha < 0$. Our state vector is defined to be $x = [x, \dot{x}]^T$, and in the cost function $z = x$ and $z_{\text{goal}} = x_{\text{goal}}$. $F_f$ is defined as the Tustin friction model, presented in Equations 3.7 to 3.12.

$$F_f = \sqrt{2e(F_{\text{brk}} - F_C)} \cdot \exp \left( -\left( \frac{\dot{x}}{v_{\text{St}}} \right)^2 \right) \cdot \frac{\dot{x}}{v_{\text{St}}} + F_C \cdot \tanh \left( \frac{\dot{x}}{v_C} \right) + f\dot{x}$$  \hspace{1cm} (3.7)

$$F_C = \mu mg$$  \hspace{1cm} (3.8)

$$f = \frac{\mu}{100}$$  \hspace{1cm} (3.9)

$$F_{\text{brk}} = 1.3F_C$$  \hspace{1cm} (3.10)

$$v_{\text{St}} = \sqrt{2}v_{\text{brk}}$$  \hspace{1cm} (3.11)

$$v_C = \frac{v_{\text{brk}}}{10}$$  \hspace{1cm} (3.12)

In Equations 3.7 to 3.12, $F_f$ is the friction force, $F_{\text{brk}}$ represents the static friction breakaway force, $F_C$ is the coulomb friction force, $v_{\text{St}}$ is the Stribeck velocity threshold, $v_C$ is the coulomb velocity threshold, and $f$ is the viscous friction coefficient. $\mu$ represents the coefficient of kinetic friction and $g$ the gravitational constant. Given the nominal parameters in Tables 3.4, the friction model produces a $F_f$ vs $\dot{x}$ curve as shown in Figure 3.2. This type of friction is notoriously difficult to compensate for in control scenarios, and gives the mass-spring-damper system a propensity to stick, then slip forward rapidly.

Table 3.4 show the mass-spring-damper parameters for the five uncertainty scenarios, as defined in Table 3.3. The disturbance on $u$ is characterized by a wandering offset and Gaussian noise with parameters as shown in Table 3.4. The offset begins at 0 and takes a step of 0.01 N with probability 0.7 at each time step. When the offset accrues to the upper end of the $u_{\text{noise}}$ range, the wandering component begins to take a step of $-0.01$ N with probability 0.7.

Tables 3.5 and 3.6 present the controller hyperparameters used in the simulated mass-spring-damper experiments. For all trials, the population size for both NHMPC algorithms was set to 300, with a horizon of 0.15s, using three knot points, with $Q = \text{diag}(1.0, 0.01)$, $Q_f = Q$ and $R = 0$. Error calculations for $\rho$, $c_3$, and $\sigma$ decay were conducted solely on $x$ with $e_{\text{max}} = 2.0$.
Results and Discussion

Five experiments with the mass-spring-damper were conducted for each NHMPC controller, one for each of the five uncertainty scenarios listed in Table 3.3. Each experiment represents 500 two-second rollouts, with the commanded goal position located at $x = 1.0$, which coincides with the surface of the wall. Given the cost function formulated in Equation 3.1 and a controller that perfectly accounts for uncertainty, one would expect the controller to drive the system rapidly close to the wall, then maintain a safe distance to prevent collisions due to system noise.

A summary of results can be seen in Figure 3.3, where ITAE is the integrated time-absolute error, a control system performance measure that penalizes solutions that wander after approaching the goal. No collisions occurred in any of the mass-spring-damper experiments. The mean ($\mu$) and standard deviation ($\sigma$) of $x$ for the 500 rollouts of each experiment can be seen in Figure 3.4 for no uncertainty, Figure 3.5 for parameter uncertainty, Figure 3.6 for input disturbance, Figure 3.7 for state measurement noise, and Figure 3.8 for the combined case.

For the mass-spring-damper experiment, NEMPC and PSOMPC demonstrate almost identical performance, showing only small differences in how far away from the wall the mass comes to rest at steady state. Overall, from Figure 3.3, one can see that PSOMPC has a lower ITAE on
average than NEMPC for uncertainty scenarios, while NEMPC maintains a lower ITAE for the ‘no uncertainty’ scenario. No collisions occurred for any experiments with the mass-spring-damper system. From Figures 3.4 to 3.8, it appears that PSOMPC experiences less variance at steady state than NEMPC, and can therefore rest closer to the wall while still maintaining a safe distance during uncertainty scenarios. However, for the ‘no uncertainty’ scenario, it can be seen that, despite having less variance than NEMPC, PSOMPC holds the system farther from the wall.

In the next section, by conducting similar experiments on a more complicated system, we will further explore the differences in behavior of NEMPC and PSOMPC in a larger optimization space, with nine dimensions for the input space instead of three.
Table 3.6: PSOMPC Hyperparameters for Mass-Spring-Damper Experiments

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho_{\text{min}}$</td>
<td>0.2</td>
</tr>
<tr>
<td>$\rho_{\text{max}}$</td>
<td>1.6</td>
</tr>
<tr>
<td>$c_{\text{sm}}$</td>
<td>0.6</td>
</tr>
<tr>
<td>$c_{\text{sm}}$</td>
<td>0.4</td>
</tr>
<tr>
<td>$n_{\text{swarms}}$</td>
<td>3</td>
</tr>
<tr>
<td>$r_{\text{valley}}$</td>
<td>65.0</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>80.0</td>
</tr>
<tr>
<td>$d_{\text{converge}}$</td>
<td>18.0</td>
</tr>
<tr>
<td>$\sigma_{\text{noise}}$</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Figure 3.3: Performance of PSOMPC and NEMPC on a constrained mass spring damper model under the five uncertainty scenarios. Data from 500 two-second rollouts are presented for NEMPC and PSOMPC in each uncertainty scenario, for a total of 5000 rollouts.
3.5.2 Three-link Robot Arm

System Definition

To explore additional complexity in a constrained simulation, we use a three-link robot in the presence of gravity. We add an obstacle to the robot’s work envelope and command it to move from a resting downward position to a balanced vertical position.

The three-link robot is comprised of three links attached with pin joints, so the dynamic equations take the following canonical form:

\[
M(q)\ddot{q} + C(q, \dot{q}) + b\dot{q} + \tau_{grav} = u + u_{noise} \tag{3.13}
\]

In Equation 3.13, \(q\) is the vector of generalized coordinates, \(M(q)\) is a configuration dependent inertia matrix, a function of our system parameters \(l, m, \) and \(I\) which represent, the link length, mass, and link mass moment of inertia about the pin joint, respectively. \(C(q, \dot{q})\) represents torques produced by centrifugal and Coriolis forces and is also a function of \(l, m, \) and \(I.\) \(b\) is a viscous damping coefficient, \(\tau_{grav}\) are the torques applied by gravity on the robot, \(u\) are the applied
torques from the motors, with $u_{\text{noise}}$ as the wandering Gaussian input disturbance. We choose as our state $x = [\dot{q}, q]^T$, and in the cost function $z = [x, y]^T$ and $z_{\text{goal}} = [x_{\text{goal}}, y_{\text{goal}}]^T$ where $x$ and $y$ are the end effector position in Cartesian space for a given joint angle $\theta$. For each simulation, each link has the same $m$, $l$, and $I$ as the other links.

We simulate the three-link robot arm in the presence of an obstacle in the upper-left portion of the robot’s workspace, where $z \in \mathcal{Z}$ if $x < -0.1$ & $y > 1.25$, as shown in Figure 3.9. This obstacle is placed such that the natural instability of the arm in its vertical position will cause frequent collisions with the end effector. In addition, as $l$ varies in the differential equations representing the dynamics of the arm, the equations for the kinematics of the arm treat the link length as constant with $l = 0.5$ meters. Allowing the kinematics of the robot to have uncertainty is an interesting problem to be solved, but is out of the scope of this paper, and should be examined in future work (especially in the case of one of our hardware applications - soft robot control).

The cost function of Equation 3.1 is defined such that $z$ represents the $x,y$ position of the end effector in the robot’s task space. $z_{\text{goal}} = [0.0, 1.5]$ is defined in task space. $Q$ and $Q_f$ are sized appropriately to penalize error in task space. Error calculations for $\rho$, $c_s$, and $\sigma$ decay were
conducted in joint space, with $e = \| q_{\text{goal}} - q \|$, with $e_{\text{max}} = \pi \sqrt{3}$. Since $q = [0, 0, 0]$ puts the end effector at $z = z_{\text{goal}}$, error calculated in joint space should be sufficient for determining NEMPC and PSOMPC decaying parameters, and requires fewer evaluations of the robot’s forward kinematics.

Table 3.7 shows the three-link robot arm parameters for the five uncertainty scenarios. The wandering $u$ disturbance is calculated with the same method as the mass-spring-damper system, albeit with the parameters shown in Table 3.7.

Tables 3.8 and 3.9 present the controller hyperparameters used in the three-link robot arm experiments. For all trials, the population size for both NHMPC algorithms was set at 500, the time horizon at 0.4s, using three knot points, with $Q = \text{diag}(1.0, 1.0)$, $Q_f = \text{diag}(10.0, 10.0)$ and $R = 0$. $Q$ and $Q_f$ were set up to penalize error in task space, as defined above.

**Results and Discussion**

As with the mass-spring-damper, five experiments with the three link robot arm were conducted with each controller, one for each of the five uncertainty scenarios. Each experiment
represents 500 two-second rollouts, with the commanded end effector goal position located at \( x = 0.0, y = 1.5 \), which is to the right of the obstacle. Given the cost function formulated in Equation 3.1 and a controller that perfectly accounts for uncertainty, one would expect the controller to drive the system rapidly to its goal position, then maintain a safe distance from the obstacle to prevent collisions due to system noise.

A summary of control performance results can be seen in Figure 3.10. Plots of the three-link end effector position over time for the 500 rollouts of each experiment can be seen in Figure 3.11 for ‘no uncertainty,’ Figure 3.12 for ‘parameter uncertainty,’ Figure 3.13 for ‘input disturbance,’ Figure 3.14 for ‘state measurement noise,’ and Figure 3.15 for the combined case. The performance in all five scenarios is relatively similar for the majority of the movement, but differs when in proximity to the goal position. With ‘no uncertainty,’ ‘input disturbance,’ and ‘state measurement error,’ the end effector seems to only wander side to side near the goal, but when parameter uncertainty is introduced in Tables 3.12 and 3.15, the end effector appears to wander vertically as well, indicating that both controllers struggle more to compensate for parameter uncertainty.
Figure 3.8: Mean and standard deviation of PSOMPC and NEMPC control of a non-linear mass spring damper with all forms of uncertainty combined. Data from 500 two-second rollouts are presented for NEMPC and PSOMPC.

When compared with NEMPC’s control performance in Figure 3.10, PSOMPC demonstrates its ability to compensate for uncertainty at the cost of decreased performance. For all scenarios, including ‘no uncertainty,’ PSOMPC is able to locate solutions with fewer opportunities for collisions, especially when parameter uncertainty is taken into account. However, from Figures 3.11 to 3.15, it is clear to see that NEMPC locates consistent solutions, whereas PSOMPC is more erratic in its solution space, resulting in decreased ITAE performance.

NEMPC primarily searches the solution space in a stochastic and combinatorial manner, allowing its single population of solutions to rapidly traverse the solution space and focus on a single valley in the cost landscape. This makes NEMPC a dynamically transparent controller, or a controller that does not add significant dynamics to the tracking of valleys. PSOMPC, on the other hand, maintains multiple swarms that each have their own dynamics, and take time to react to a drastic change in the cost landscape. The assumptions that the swarms allow us to make about relationships between solutions are critical for incorporating uncertainty robustness, but they also cause PSOMPC to not be as dynamically transparent as NEMPC. The particle motion in PSOMPC has a significant impact on its performance, and is hard to evaluate in a multidimensional optimiza-
tion space such as the three link robot arm control problem. A generalized set of hyperparameters that provide good PSOMPC performance may be difficult to identify, but will be a necessary direction of future research.

An interesting result from Figure 3.10 is that the ITAE and number of collisions for the ‘state’ uncertainty scenario is lower than ‘no uncertainty’ for both controllers. This suggests that artificially adding state measurement uncertainty inside PSOMPC or NEMPC may increase the robustness of either algorithm to uncertainty.

We next examine the effects of the uncertainty robustness capabilities of PSOMPC, comparing three-link performance and collisions for various values of $\sigma_{\text{noise}}$ and $n_{\text{swarms}}$, as defined in Table 3.2. Figure 3.16 shows a summary of control performance for PSOMPC with varying values of $\sigma_{\text{noise}}$, but the remainder of the parameters were set at the nominal values defined in Table 3.9. Figure 3.17 does the same, albeit while varying the $n_{\text{swarms}}$ parameter. For both plots, the red dataset represents the nominal parameters used in the previous three-link robot arm experiments as shown in Figures 3.10 to 3.15.
Table 3.7: Three-Link Robot Arm Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g ) ((m/s^2))</td>
<td>9.81</td>
</tr>
<tr>
<td>( x_{\text{wall}} ) ((m))</td>
<td>-0.1</td>
</tr>
<tr>
<td>( y_{\text{wall}} ) ((m))</td>
<td>1.25</td>
</tr>
<tr>
<td>( \beta )</td>
<td>10.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Nominal</th>
<th>Min</th>
<th>Max</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m ) ((kg))</td>
<td>0.5</td>
<td>0.3</td>
<td>0.7</td>
<td>parameter</td>
</tr>
<tr>
<td>( b ) ((Ns/rad))</td>
<td>0.05</td>
<td>0.03</td>
<td>0.07</td>
<td>parameter</td>
</tr>
<tr>
<td>( I )</td>
<td>0.1</td>
<td>0.06</td>
<td>0.14</td>
<td>parameter</td>
</tr>
<tr>
<td>( l ) ((m))</td>
<td>0.5</td>
<td>0.4</td>
<td>0.6</td>
<td>parameter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Nominal</th>
<th>Mean</th>
<th>Std. Dev</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_{\text{meas}} ) ((rad))</td>
<td>0.0</td>
<td>0.0</td>
<td>0.01</td>
<td>state</td>
</tr>
<tr>
<td>( \dot{q}_{\text{meas}} ) ((rad/s))</td>
<td>0.0</td>
<td>0.0</td>
<td>0.01</td>
<td>state</td>
</tr>
<tr>
<td>( u_{\text{noise}} ) ((Nm))</td>
<td>0.0</td>
<td>[-0.2, 0.2]</td>
<td>1.0</td>
<td>input</td>
</tr>
</tbody>
</table>

Table 3.8: NEMPC Hyperparameters for Three-Link Experiments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma_{\text{noise}} )</td>
<td>66.29</td>
</tr>
<tr>
<td>( n_{\text{parents}} )</td>
<td>12</td>
</tr>
<tr>
<td>( n_{\text{strangers}} )</td>
<td>7</td>
</tr>
<tr>
<td>( p_{\text{mutate}} )</td>
<td>0.94</td>
</tr>
</tbody>
</table>

An analysis of Figure 3.16 shows the benefits and drawbacks of adding simulation noise to the PSOMPC algorithm. As more simulation noise is added, the controller is able to avoid collisions more effectively, at the cost of worse ITAE performance. There is a point, however, where diminishing returns are seen, and more simulation noise does little to reduce collisions, while still worsening the system’s performance.

In Figure 3.17, we see the tradeoffs associated with increasing the number of independent swarms tracked in the PSOMPC algorithm. A single swarm results in very poor ITAE performance, and any reduction in collisions is likely only due to the controller never getting the end effector close enough to the goal position to incur a collision. However, with more than three swarms, the
Table 3.9: PSOMPC Hyperparameters for Three-Link Experiments

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho_{\text{min}}$</td>
<td>0.37</td>
</tr>
<tr>
<td>$\rho_{\text{max}}$</td>
<td>0.20</td>
</tr>
<tr>
<td>$c_{\text{smin}}$</td>
<td>0.42</td>
</tr>
<tr>
<td>$c_{\text{smam}}$</td>
<td>1.88</td>
</tr>
<tr>
<td>$n_{\text{swarms}}$</td>
<td>5</td>
</tr>
<tr>
<td>$r_{\text{valley}}$</td>
<td>2.09</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>88.85</td>
</tr>
<tr>
<td>$d_{\text{converge}}$</td>
<td>82.59</td>
</tr>
<tr>
<td>$\sigma_{\text{noise}}$</td>
<td>7.96</td>
</tr>
</tbody>
</table>

underlying tradeoff becomes more obvious. As the number of swarms increases, PSOMPC is able to find more robust solutions, at the cost of worse ITAE performance.

Better formulations of uncertainty modeling could reduce the negative effect on ITAE performance, and should be a topic of future studies. However, the tradeoff between ITAE performance and robustness to uncertainty is significant. Each system will likely have a different point of diminishing returns, and there may be interactions with other hyperparameters that will be necessary to explore, therefore more research into the PSOMPC hyperparameters is recommended. For the purposes of this thesis, however, Figures 3.16 and 3.17 demonstrate that simulation noise and multiple swarms are key elements of the PSOMPC algorithm, and account for much of the uncertainty robustness seen in the algorithm.

3.6 Soft Robot Experiments

3.6.1 System Definition

The hardware system used for this work is a continuum joint comprised of four extensible, pressurized chambers centered around an inextensible steel cable, as shown in Figure 3.18. A pressure differential in these four chambers causes the joint to bend in two axes. We use the same singularity-free kinematic relationships derived in [67] where the curvature of the continuum joint is parameterized as two separate rotations ($u$ and $v$) about orthogonal axes ($x$ and $y$), which lie at
Figure 3.10: Performance of PSOMPC and NEMPC on a constrained three link robot arm model under various conditions of system uncertainty. Data from 500 rollouts are presented for NEMPC and PSOMPC in each uncertainty scenario, representing 5000 rollouts in total.

In Equation 3.14, $\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \mathbf{τ}$

In Equation 3.14, $M(q)$ is the mass matrix, $C(q, \dot{q})$ is the Coriolis matrix, $g(q)$ is a vector of torques caused by gravity, $q(t) = [\theta, \phi]^T$ is a vector of generalized coordinates, and $\tau$ is a vector of generalized forces.

For notational clarity in this paper, we define $\theta = u$ and $\phi = v$. The dynamics are presented in Equation 3.14.
Figure 3.11: Angle data for the three link robot arm experiments with no uncertainty in the system. This figure contains data from 500 rollouts for each algorithm.

The pressures in each chamber are modeled as a first-order response

\[ \dot{p}(t) = A(p_{\text{ref}}(t) - p(t)) \]  

(3.15)

In Equation 3.15 \( p(t) \) is a vector of pressures, \( p_{\text{ref}}(t) \) is a vector of reference (i.e. commanded) pressures, and \( A \) is a diagonal matrix of coefficients representing the fill/vent rate of the pneumatic valves. Numerical values for the parameters used in this model are discussed in [30].

There are several effects from material properties that are not accounted for in Equation 3.14, but can be included in the vector of generalized forces \( \tau \). We include these effects as a linear spring term \( K_{\text{spring}}q \), where \( K_{\text{spring}} \) is a diagonal matrix, which pulls the joint towards a completely vertical configuration, and a viscous damping term \( K_{d}\dot{q} \), where \( K_{d} \) is also a diagonal matrix. The pressure-to-torque mapping term \( K_{\text{prs}}p \) maps pressure differentials in each antagonistic pair of bellows to a torque about each axis where bending in \( \phi \) and \( \theta \) occur. These additions, coupled with Equation 3.15, result in our final analytical dynamic model:

\[ M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = K_{\text{prs}}p - K_{d}\dot{q} - K_{\text{spring}}q \]  

(3.16)
Figure 3.12: Path data for PSOMPC and NEMPC control of a three link robot arm with varying parameters. This figure contains data from 500 rollouts for each algorithm.

We choose as our state $x = [p, \dot{q}, q]^T$, and in the cost function $z = x$ and $z_{\text{goal}} = z_{\text{goal}}$.

Following the methods of [30], a deep neural net (DNN) was trained to approximate these dynamic equations, then fine-tuned on data gathered from the hardware system to approximate the real dynamics more accurately. That DNN is used as the internal simulation model for both NEMPC and PSOMPC throughout the experiments conducted on the soft robot continuum joint.

For the experiments conducted on the continuum joint, a spherical tip was added to the robot, and an obstacle placed roughly at the $\phi = 0, \theta = -0.4$ position, as pictured in 3.18 and diagrammed in 3.19. Two experiments were conducted, one in simulation and one on the actual hardware. For both experiments, the robot was commanded to move from one end of its workspace to the opposite side, where a direct path would involve a collision with the obstacle in its workspace (similar to the three link robot arm experiment). For both experiments, NEMPC and PSOMPC had the DNN approximation of hardware dynamics obtained in [30] as the controller’s model of the plant. For the simulated experiment, that same DNN was also used as the plant which received commands generated by NEMPC and PSOMPC. This demonstrated performance where the controller had a perfect model of the plant, similar to the ‘no uncertainty’ case for the simulation.
Figure 3.13: Path data for PSOMPC and NEMPC control of a three link robot arm with wandering noise on the system input. This figure contains data from 500 rollouts for each algorithm.

experiments in Section 3.5. For the hardware experiment, the plant was the robot itself, receiving commands through the Robot Operating System (ROS).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\text{noise}}$</td>
<td>10</td>
</tr>
<tr>
<td>$n_{\text{parents}}$</td>
<td>50</td>
</tr>
<tr>
<td>$n_{\text{strangers}}$</td>
<td>10</td>
</tr>
<tr>
<td>$p_{\text{mutate}}$</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Tables 3.10 and 3.11 present the controller hyperparameters used in the soft robot continuum joint experiments.

For both experiments, the population size for both NHMPC algorithms was set at 300, the time horizon at 0.2s, using three knot points, with $Q = \text{diag}(0, 0, 0, 0, 0, 1.0, 1.0)$, $Q_f = Q$ and $R = \text{diag}(0.000001, 0.000001, 0.000001, 0.000001)$. This results in weighting error in $\phi$ and $\theta$, as well as a slight weight on $\Delta u$. In addition, $\Delta u$ was subjected to a slew rate constraint outside
Figure 3.14: Path data for PSOMPC and NEMPC control of a three link robot arm with noise applied to state (position and velocity) measurements. This figure contains data from 500 rollouts for each algorithm.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho_{\text{min}}$</td>
<td>0.57</td>
</tr>
<tr>
<td>$\rho_{\text{max}}$</td>
<td>0.61</td>
</tr>
<tr>
<td>$c_{\text{sm}}$</td>
<td>0.13</td>
</tr>
<tr>
<td>$c_{\text{sm}}$</td>
<td>1.12</td>
</tr>
<tr>
<td>$n_{\text{swarms}}$</td>
<td>2</td>
</tr>
<tr>
<td>$r_{\text{valley}}$</td>
<td>31.9</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>62.9</td>
</tr>
<tr>
<td>$d_{\text{converge}}$</td>
<td>70.85</td>
</tr>
<tr>
<td>$\sigma_{\text{noise}}$</td>
<td>0.01</td>
</tr>
</tbody>
</table>

of the controller, restricting $\Delta u$ to 1 kPa/(time step) for the hardware experiment and 2 kPa/(time step) for the simulation experiment. This was necessary to prevent chaotic pressure changes that cause the hardware to behave erratically, and was placed outside the controller to still give the controller the freedom to explore the entire control space. Control commands were sent at 50 Hz. Each experiment consisted of 50 trials for each controller, with each trial lasting 6 seconds. The
Figure 3.15: Path data for PSOMPC and NEMPC control of a three link robot arm with all forms of uncertainty combined. This figure contains data from 500 rollouts for each algorithm.

The simulated soft robot continuum joint results are presented in Figures 3.20 and 3.21. The obstacle displayed in these figures is at the same location as the hardware experiment in the next section, located approximately at $\phi = 0$ and $\theta = -0.4$, but exactly positioned as calibrated in the hardware experiment.

From these figures, we see that with a perfect model, PSOMPC and NEMPC perform well for soft robot control, and generate smooth trajectories. As in the simulation experiments in Section 3.5, we see in Figure 3.21 that NEMPC finds consistent solutions, while PSOMPC varies more in its solutions. However, we see in Figure 3.20 that PSOMPC actually experiences more collisions with the simulated obstacle than NEMPC. However, we will see in the next section that, when a model that contains significant uncertainty relative to the real system is used, PSOMPC violates constraints less frequently than NEMPC.
Figure 3.16: Performance of PSOMPC on a constrained three link robot arm model under various conditions of system uncertainty. For each value of $\sigma_{\text{noise}}$ shown in the legend, 500 two-second experiments were conducted in the same manner as those in Figure 3.10, for a total of 20,000 rollouts.

3.6.3 Hardware Results and Discussion

The hardware experiment results are presented in Figures 3.22 and 3.24. For this experiment, ‘Collisions’ as reported in Figure 3.22 are physical collisions with the obstacle, not collisions with the software representation of the obstacle, therefore a line crossing the ‘Barrier’ in Figure 3.24 does not necessarily indicate a collision. The software representation of the obstacle was increased in diameter because, on hardware, extra robustness to uncertainty was needed for both controllers to perform well.

In the 50 rollouts that were conducted for each controller, PSOMPC never touched the physical obstacle, while NEMPC collided in 15 of the 50 rollouts, as seen in Figure 3.22. Figure 3.23 presents the maximum depth with which NEMPC and PSOPMPC penetrated the boundary of the obstacle as defined in software. One can see that, although both algorithms violate the virtual constraint many times, NEMPC does so with greater frequency and severity. However, as with the simulation experiments in Section 3.5, PSOMPC reduced the number and severity of collisions at the cost of poorer ITAE performance. In addition, PSOMPC’s propensity to take more varied
Figure 3.17: Performance of PSOMPC on a constrained three link robot arm model under various conditions of system uncertainty. For each value of $n_{\text{swarms}}$ shown in the legend, 500 two-second experiments were conducted in the same manner as those in Figure 3.10. Note that increasing the number of swarms does not increase the size of PSOMPC’s population, it merely splits the total population into more individual groups. For each value shown in the legend, 500 two-second experiments were conducted in the same manner as those in Figure 3.10, for a total of 20,000 rollouts.

Solutions resulted in more erratic paths when run on a physical robot, as seen in Figure 3.24. NEMPC, true to previous results, took more consistent paths that approached the goal faster, with the notable exception of several trials that swung far to the side of the obstacle. In addition, the slew rate constraint on $\Delta u$ occasionally causes NEMPC to fail to react in time to prevent a collision.

Of additional note is the jerky nature of the PSOMPC solutions. These wavy patterns decrease the speed with which the robot moves, and are likely due to hardware dynamics that were not adequately learned when training the DNN to approximate the robot’s performance (see [30]). With more research into learning and approximating hardware dynamics, it could be expected that these wavy solutions should become smooth. Overall, however, PSOMPC handles constraints in the presence of uncertainty effectively in hardware as well as in simulation.
3.7 Conclusion

This paper has introduced particle swarm model predictive control (PSOMPC) and evaluated its performance against nonlinear evolutionary model predictive control (NEMPC) for control of various robotic systems in the presence of uncertainty. Introducing variation (seen as uncertainty) in the state seems to help both controllers perform more robustly with respect to collisions. Future work should examine the possibility of introducing state uncertainty in the simulation portion of the controller and its effect on the controller’s robustness to uncertainty.

Cost function formulation with a simple penalty on collisions was shown to be enough to encourage robust behavior. However, more advanced cost functions should be evaluated in future work to determine cost function guidelines to handle constrained scenarios in the presence of uncertainty. Cost functions that penalize proximity to obstacles or that penalize wandering in the system could be considered. Of particular interest would be a cost function that considers the shape of an uncertainty distribution at the end effector of a multi-link robot arm, configuring the arm to orient the distribution to have minimal impact on the robot’s performance.

Several types of uncertainty were explored, but uncertainty in robot kinematics will be a necessary topic of research, as soft robots are used to handle payloads. As most soft robots handle
Figure 3.19: A diagram of the experimental setup for the soft robot continuum joint. A hanging obstacle lies within the robot’s workspace and is reachable with its spherical tip. Contact between the obstacle and spherical tip is considered a collision.

In this work, we have demonstrated that particle swarm model predictive control (PSOMPC), a new variant of nonlinear heuristic model predictive control (NHMPC), performs favorably in conditions where uncertainty is present in the system to be controlled. When compared with nonlinear evolutionary model predictive control (NEMPC), PSOMPC demonstrates a better robustness to uncertainty, reducing collisions for the three-link robot arm by 71%. Performance of both controllers
Figure 3.20: Performance of PSOMPC and NEMPC for 50 rollouts of an obstacle approach task on a simulated soft robot continuum joint.

was demonstrated on a simulated nonlinear mass-spring-damper, a simulated three-link robot arm, and a soft robot continuum joint in hardware. A naive cost function was used to penalize collisions with an object in each system’s workspace, but proved effective enough to encourage each controller to maintain some distance from the obstacle to account for uncertainty.
Figure 3.21: $\phi$ and $\theta$ history from the 50 rollouts of the simulated soft robot continuum joint experiment.

Figure 3.22: Performance of PSOMPC and NEMPC for an obstacle approach task on the soft robot continuum joint. Note that the ‘barrier’ as recorded in software is slightly larger than the obstacle in hardware, therefore a ‘collision’ in software does not always coincide with a collision in hardware. This figure reports physical collisions, not software collisions. Data from 50 rollouts for each algorithm is presented here.
Figure 3.23: A histogram of the maximum penetration into the obstacle region as defined in software. Both NEMPC and PSOMPC crossed the barrier boundary as defined in software multiple times, but NEMPC penetrates the obstacle farther and with a greater frequency.

Figure 3.24: $\phi$ and $\theta$ history from the 50 rollouts of the hardware experiment. The software ‘barrier’ is larger than the obstacle in hardware, therefore crossing the red dashed line is not necessarily indicative of a collision in hardware. Data from 50 rollouts for each algorithm is presented here.
CHAPTER 4. CONCLUSIONS AND FUTURE WORK

Throughout this thesis, two forms of nonlinear heuristic model predictive control (NHMPC) were developed. Both nonlinear evolutionary model predictive control (NEMPC) and its variant, particle swarm model predictive control (PSOMPC) have been developed and analyzed on simulated and hardware systems. Both algorithms have demonstrated potential for control of soft robots as a testbed for the control of uncertain high-DoF robots. PSOMPC has shown promising preliminary results in its ability to choose solutions robust to uncertainty. As part of collaborative journal paper where three students were involved, a better method for approximating hardware dynamics using a DNN was presented. In addition, the Rad Models package was introduced as a companion to NEMPC and PSOMPC functionality.

These contributions should serve as a foundation for future work in this area, enabling better control and easier interfaces with a wide variety of systems. However, there are several areas where more work could yield better results with careful planning and direction as outlined below.

4.1 Future Work

4.1.1 RaD Models

The RaD Models package with its DNN approximation module finds its greatest utility when used with PSOMPC and NEMPC for simulated experiments. However, the package requires external tracking of states when running a simulation outside of NEMPC or PSOMPC. An object oriented wrapper could be developed to track a system’s states, accepting inputs and time information to continue to simulate a system without requiring a user to track the system state.
4.1.2 DNN Modeling

In Chapter 2, the need for future work in DNN approximations was discussed. Future work could improve DNN performance by using a buffer of the last $n$ states. This time sequence data could be leveraged by a fully-connected network, or a recurrent neural network (RNN). In addition, it would be worthwhile to conduct a search into how sensitive the model is to the frequency content in the data used for training. Chapter 2 presented evidence that using square waves to explore and learn the state space is an efficient method because the trained models generalized relatively well to sine waves and ramps, but that learning sine waves and ramps did not generalize well to prediction with square waves.

Future work could also include a learning approach which allows the platform to continuously learn an error model online. In addition, exploring the state space randomly to gather training data is not always possible on some hardware platforms. Future work could include an exploration of how learning from a safe subspace of the state space can generalize to control over the entire reachable state space.

4.1.3 Dynamic Particle Swarm MPC

In Chapter 3, future work directions for PSOMPC were discussed. Future work should explore the effects of PSOMPC hyperparameters on particle motion to identify which hyperparameters are sensitive and which can remain at a constant value for most systems. In addition, more advanced cost functions should be evaluated in future work to determine cost function guidelines to handle constrained scenarios in the presence of uncertainty. Future work could explore cost functions that:

- penalize proximity to obstacles
- penalize variations in the control input that cause erratic robot behavior
- consider the shape of an uncertainty distribution at the end effector of a multi-link robot arm, configuring the arm to orient the distribution to have minimal impact on the robot’s performance
From the experiments in Chapter 3, state uncertainty seems to help the controllers perform more robustly with respect to collisions, and should be explored as a method of introducing uncertainty robustness in a system. Uncertainty in robot kinematics will also be a necessary topic of research, as soft robots are used to handle payloads and perform real-world tasks.

4.2 Conclusion

In spite of the amount of future work to be conducted, this thesis presents key improvements to the NEMPC algorithm, and a new control method that shows promise in the control of robotic systems in the presence of uncertainty. In review, the contributions of this thesis are as follows:

1. Development of particle swarm model predictive control (PSOMPC), a variant of NEMPC utilizing the particle swarm optimization routine and incorporating hyperparameters for the modeling of system uncertainty

2. A second revision of NEMPC, with parallelized mating and mutation for a large decrease in computation time

3. A method for obtaining better DNN approximations of hardware systems

4. The RaD Models package, a collection of system models with a common interface for use with model-based controllers

5. The DNN Approximation module inside the RaD Models package, a module that abstracts away the need for expertise in DNN architecture and allows for DNN training and initialization with a simple interface
REFERENCES


of the under-actuated nonlinear acrobot.” In *2014 IEEE 13th International Workshop on Advanced Motion Control (AMC)*, IEEE, pp. 464–469.


[56] Desaraju, V. Safe, efficient, and robust predictive control of constrained nonlinear systems.


APPENDIX A. DETAILS OF PARALLELIZATION TO IMPROVE EXECUTION TIME FOR NEMPC

Nonlinear heuristic model predictive control (NHMPC) and its variants, nonlinear evolutionary model predictive control (NEMPC) and particle swarm model predictive control (PSOMPC), were sped up via the elimination of extraneous For loops. To do so, we leveraged the Python Numpy library’s capabilities of ‘broadcasting,’ where large data structures are operated on with one line of Python code, speeding up execution by orders of magnitude (see Figure 2.1). This is referred to as ‘vectorizing.’

To better understand the work required in vectorizing the NHMPC framework and its variants, one first must understand the Python Numpy library and its pitfalls. Figure A.1 demonstrates the most common pitfall when utilizing Numpy, trying to use arrays of different shapes in the same operation.

```
a.shape = (2, 1)
b.shape = (2,)
(a + b).shape = (2, 2)
```

Figure A.1: An example of Numpy’s broadcasting pitfalls.

When array \( a \) and \( b \) are created, both have only two elements or data points, and one might assume that adding the two would result in another array with two elements, but Numpy’s broadcasting rules result in an array of shape \( (2, 2) \), which is often undesirable. To properly utilize broadcasting, it helps to visualize data structures as ‘buckets’ of data, as seen in Figure A.2. Colored stacks of blocks indicate individual arrays, organized in buckets or bins in arrays \( a \) and \( b \).
When performing an operation utilizing $a$ and $b$, $b$ will broadcast its data to match the shape of $a$, as seen in Figure A.3. The trailing dimensions are lined up (4, in this case), and any missing dimensions are broadcast to be the same size as the other array.

‘Singleton’ dimensions also broadcast, as seen in Figures A.4 and A.5. When trailing dimensions are lined up, dimensions of unit 1 are broadcast, and data copied to match the sizes of the two arrays.

Utilizing these broadcasting rules, we can set up better data structures for $P$ and other variables within NHMPC, NEMPC, and PSOMPC, as seen in Figure A.6. When operations occur
between these large data structures, Numpy performs the computations in a wrapped C++ library, greatly speeding up the execution times as seen in Figure 2.1. In addition, many of the operations become more intuitive and clean when reading the code.

The vectorization of NHMPC and its variants was a key contribution of this thesis, and an understanding of the underlying principles can enable faster Python implementations of future controllers, especially when utilizing heuristic optimization routines. Most, if not all heuristic optimization routines require operations performed on each member of the population or swarm, and the ability to perform these computations in block operations will greatly speed up their execution.
Figure A.6: The shapes of the data structures used in NHMPC.